

siunitx – A comprehensive (SI) units package*

Joseph Wright[†]

Released 2019-11-02

Contents

I	siunitx – Overall set up	1
1	siunitx implementation	1
1.1	Initial set up	1
1.2	Safety checks	1
1.3	Top-level scratch space	2
1.4	User interfaces	2
1.4.1	Preamble commands	2
1.4.2	Document commands	3
1.5	Document commands in bookmarks	5
1.6	“Glue” commands	6
1.7	Table column	6
II	siunitx-angle – Formatting angles	8
1	Formatting angles	8
1.1	Key-value options	8
2	siunitx-angle implementation	9
III	siunitx-quantity – Multi-part numbers	16
1	siunitx-compound implementation	16
1.1	Lists	16
1.2	Products	17
1.3	Ranges	18
1.4	Standard settings for module options	19
IV	siunitx-number – Parsing and formatting numbers	21

*This file describes v3.0.0-alpha.2, last revised 2019-11-02.

[†]E-mail: joseph.wright@morningstar2.co.uk

1	Formatting numbers	21
1.1	Key-value options	22
2	siunitx-number implementation	25
2.1	Initial set-up	25
2.2	Main formatting routine	25
2.3	Parsing numbers	26
2.4	Processing numbers	40
2.5	Formatting parsed numbers	55
2.6	Miscellaneous tools	62
2.7	Messages	63
2.8	Standard settings for module options	63
V	siunitx-print – Printing material with font control	65
1	Printing quantities	65
1.1	Key-value options	65
2	siunitx-print implementation	67
2.1	Initial set up	68
2.2	Printing routines	69
2.3	Standard settings for module options	76
VI	siunitx-quantity – Quantities	78
1	siunitx-quantity implementation	78
1.1	Initial set-up	78
1.2	Main formatting routine	78
1.3	Standard settings for module options	80
1.4	Adjustments to units	80
VII	siunitx-symbol – Symbol-related settings	81
1	siunitx-symbol implementation	81
1.1	Bookmark definitions	84
VIII	siunitx-table – Formatting numbers in tables	86
1	Numbers in tables	86
1.1	Key-value options	86

2	siunitx-table implementation	87
2.1	Interface functions	87
2.2	Collecting tokens	88
2.3	Separating collected material	90
2.4	Printing numbers in cells: spacing	91
2.5	Printing just text	93
2.6	Number alignment: core ideas	93
2.7	Directly printing without collection	96
2.8	Printing numbers in cells: main functions	98
2.9	Standard settings for module options	105
IX	siunitx-unit – Parsing and formatting units	106
1	Formatting units	106
2	Defining symbolic units	107
3	Per-unit options	108
4	Units in (PDF) strings	108
5	Pre-defined symbolic unit components	109
5.1	Key-value options	112
6	siunitx-unit implementation	114
6.1	Initial set up	114
6.2	Defining symbolic unit	115
6.3	Applying unit options	117
6.4	Non-standard symbolic units	118
6.5	Main formatting routine	119
6.6	Formatting literal units	120
6.7	(PDF) String creation	123
6.8	Parsing symbolic units	123
6.9	Formatting parsed units	128
6.10	Non-Latin character support	138
6.11	Pre-defined unit components	138
6.12	Messages	142
6.13	Standard settings for module options	142
X	siunitx-abbreviations – Abbreviations	144
1	siunitx-abbreviation implementation	146
XI	siunitx-command – Units as document command	150
1	Creating units as document commands	150
1.1	Key-value options	150

2	siunitx-command implementation	150
2.1	Options	151
2.2	Creation of unit document commands	151
2.3	Standard settings for module options	153
XII	siunitx-emulation – Emulation	154
1	siunitx-emulation implementation	154
1.1	Version 2	154
1.2	Preamble commands	154
1.3	Document commands	154
1.4	Angle options	155
1.5	Combination functions options	156
1.6	Command options	156
1.7	Print options	156
1.8	Number options	159
1.8.1	Table options	161
1.9	Unit options	164
	Index	166

Part I

siunitx — Overall set up

1 siunitx implementation

Start the DocStrip guards.

```
1 <*package>
    Identify the internal prefix (LATEX3 DocStrip convention).
2 <@=siunitx>
```

1.1 Initial set up

Load only the essential support (expl3) “up-front”.

```
3 \RequirePackage{expl3}
    Make sure that the version of l3kernel in use is sufficiently new. This will also trap
    any problems with l3packages (as the two are now tied together, version-wise).
4 \@ifpackagelater {expl3}{2018-06-01}
5 {}
6 {%
7   \PackageError{siunitx} {Support package expl3 too old}
8   {%
9     You need to update your installation of the bundles 'l3kernel' and
10    'l3packages'.\MessageBreak
11    Loading~siunitx~will~abort!%
12   }%
13   \endinput
14 }%
    Identify the package and give the over all version information.
15 \ProvidesExplPackage {siunitx} {2019-11-02} {3.0.0-alpha.2}
16 {A comprehensive (SI) units package}
```

1.2 Safety checks

`__siunitx_load_check:` There are a number of packages that are incompatible with siunitx as they cover the same concepts and in some cases define the same command names. These are all tested at the point of loading to try to trap issues, and a couple are also tested later as it's possible for them to load without an obvious error if siunitx was loaded first.

The testing here is done in a group so that the tests do not add anything to the hash table.

```
17 \msg_new:nnnn { siunitx } { incompatible-package }
18 { Package~'#1'~incompatible. }
19 { The~#1~package~and~siunitx~are~incompatible. }
20 \cs_new_protected:Npn __siunitx_load_check:n #1
21 {
22   \group_begin:
23   \@ifpackageloaded {#1}
24   {
25     \group_end:
```

```

26         \msg_error:nxx { siunitx } { incompatible-package } {#1}
27     }
28     { \group_end: }
29 }
30 \clist_map_function:nN
31 { SIunits , sistyle , unitsdef , fancyunits }
32 \__siunitx_load_check:n
33 \AtBeginDocument
34 {
35     \clist_map_function:nN { SIunits , sistyle }
36     \__siunitx_load_check:n
37 }

```

(End definition for __siunitx_load_check:.)

1.3 Top-level scratch space

\l__siunitx_tmp_tl Scratch space for the interfaces.

```

38 \tl_new:N \l__siunitx_tmp_tl

```

(End definition for \l__siunitx_tmp_tl.)

1.4 User interfaces

The user interfaces are defined in terms of documented code-level ones. This is all done here, and will appear in the .sty file before the relevant code. Things could be rearranged by DocStrip but there is no advantage.

User level interfaces are all created by xparse

```

39 \RequirePackage { xparse }

```

1.4.1 Preamble commands

```

\DeclareSIPower Pass data to the code layer.
\DeclareSIPrefix
\DeclareSIQualifier
\DeclareSIUnit
40 \NewDocumentCommand \DeclareSIPower { +m +m m }
41 {
42     \siunitx_declare_power:NNn #1 #2 {#3}
43 }
44 \NewDocumentCommand \DeclareSIPrefix { +m m m }
45 {
46     \siunitx_declare_prefix:Nnn #1 {#2} {#3}
47 }
48 \NewDocumentCommand \DeclareSIQualifier { +m m }
49 {
50     \siunitx_declare_qualifier:Nn #1 {#2}
51 }
52 \NewDocumentCommand \DeclareSIUnit { o +m m }
53 {
54     \IfNoValueTF {#1}
55     { \siunitx_declare_unit:Nn #2 {#3} }
56     { \siunitx_declare_unit:Nnn #2 {#3} {#1} }
57 }

```

(End definition for \DeclareSIPower and others. These functions are documented on page ??.)

1.4.2 Document commands

`\qty`

```
58 \NewDocumentCommand \qty { 0 { } m m }
59 {
60   \mode_leave_vertical:
61   \group_begin:
62     \siunitx_unit_options_apply:n {#3}
63     \keys_set:nn { siunitx } {#1}
64     \siunitx_quantity:nn {#2} {#3}
65   \group_end:
66 }
```

(End definition for `\qty`. This function is documented on page ??.)

`\ang` All of a standard form: start a paragraph (if required), set local key values, do the
`\num` formatting, print the result.

```
\unit 67 \NewDocumentCommand \ang { 0 { } > { \SplitArgument { 2 } { ; } } m }
68 {
69   \mode_leave_vertical:
70   \group_begin:
71     \keys_set:nn { siunitx } {#1}
72     \_siunitx_angle:nnn #2
73   \group_end:
74 }
75 \NewDocumentCommand \num { 0 { } m }
76 {
77   \mode_leave_vertical:
78   \group_begin:
79     \keys_set:nn { siunitx } {#1}
80     \siunitx_number_format:nN {#2} \l__siunitx_tmp_tl
81     \siunitx_print:nV { number } \l__siunitx_tmp_tl
82   \group_end:
83 }
84 \NewDocumentCommand \unit { 0 { } m }
85 {
86   \mode_leave_vertical:
87   \group_begin:
88     \siunitx_unit_options_apply:n {#2}
89     \keys_set:nn { siunitx } {#1}
90     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
91     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
92   \group_end:
93 }
```

(End definition for `\ang`, `\num`, and `\unit`. These functions are documented on page ??.)

`\qtylist` Interfaces for compound values.

```
\numlist 94 \NewDocumentCommand \qtylist { 0 { } > { \SplitList { ; } } m m }
\qtyproduct 95 {
\numproduct 96   \mode_leave_vertical:
\qtyrange 97   \group_begin:
98     \siunitx_unit_options_apply:n {#3}
99   %     \keys_set:nn { siunitx } {#1}
```

```

100      % ???
101      \group_end:
102    }
103 \NewDocumentCommand \numlist { 0 { } } > { \SplitList { ; } } m }
104 {
105   \mode_leave_vertical:
106   \group_begin:
107     \keys_set:nn { siunitx } {#1}
108     \siunitx_number_list:n {#2}
109   \group_end:
110 }
111 \NewDocumentCommand \qtyproduct { 0 { } } > { \SplitList { x } } m m }
112 {
113   \mode_leave_vertical:
114   \group_begin:
115     \siunitx_unit_options_apply:n {#3}
116 %     \keys_set:nn { siunitx } {#1}
117   % ???
118   \group_end:
119 }
120 \NewDocumentCommand \numproduct { 0 { } } > { \SplitList { x } } m }
121 {
122   \mode_leave_vertical:
123   \group_begin:
124     \keys_set:nn { siunitx } {#1}
125     \siunitx_number_product:n {#2}
126   \group_end:
127 }
128 \NewDocumentCommand \qtyrange { 0 { } } m m m }
129 {
130   \mode_leave_vertical:
131   \group_begin:
132     \siunitx_unit_options_apply:n {#4}
133 %     \keys_set:nn { siunitx } {#1}
134   % ???
135   \group_end:
136 }
137 \NewDocumentCommand \numrange { 0 { } } m m }
138 {
139   \mode_leave_vertical:
140   \group_begin:
141     \keys_set:nn { siunitx } {#1}
142     \siunitx_number_range:nn {#2} {#3}
143   \group_end:
144 }

```

(End definition for \qtylist and others. These functions are documented on page ??.)

\tablenum Slightly odd set up at present: we have to have the \ignorespaces.

```

145 \NewDocumentCommand \tablenum { 0 { } } m }
146 {
147   \mode_leave_vertical:
148   \group_begin:
149     \keys_set:nn { siunitx } {#1}

```



```

150     \siunitx_cell_begin:w
151     \ignorespaces #2
152     \siunitx_cell_end:
153     \group_end:
154 }

```

(End definition for \tablenum. This function is documented on page ??.)

\sisetup A very thin wrapper.

```

155 \NewDocumentCommand \sisetup { m }
156 { \keys_set:nn { siunitx } {#1} }

```

(End definition for \sisetup. This function is documented on page ??.)

1.5 Document commands in bookmarks

In bookmarks, the siunitx document commands need to produce simple strings that represent their input as far as possible.

```

\__siunitx_bookmark_cmd:Nn To keep things fast, expandable versions of the document command are created only
    \qtyInBookmark once.
    \angInBookmark 157 \cs_new_protected:Npn \__siunitx_bookmark_cmd:Nnn #1#2#3
    \numInBookmark 158 {
    \unitInBookmark 159     \exp_args:Nc \DeclareExpandableDocumentCommand
    \numlistInBookmark 160     { \cs_to_str:N #1 ~ ( pdfstring ~ context ) }
    \qtylistInBookmark 161     {#2} {#3}
    \numproductInBookmark 162 }
    \qtyproductInBookmark 163 \__siunitx_bookmark_cmd:Nnn \qty { o m m } { #2 ~ #3 }
    \numrangeInBookmark 164 \__siunitx_bookmark_cmd:Nnn \ang { m } { }
    \qtyrangeInBookmark 165 \__siunitx_bookmark_cmd:Nnn \num { o m } { #2 }
    166 \__siunitx_bookmark_cmd:Nnn \unit { o m } { #2 }
    167 \__siunitx_bookmark_cmd:Nnn \numlist { o m } { }
    168 \__siunitx_bookmark_cmd:Nnn \qtylist { o m m } { }
    169 \__siunitx_bookmark_cmd:Nnn \numproduct { o m } { }
    170 \__siunitx_bookmark_cmd:Nnn \qtyproduct { o m m } { }
    171 \__siunitx_bookmark_cmd:Nnn \numrange { o m m } { }
    172 \__siunitx_bookmark_cmd:Nnn \qtyrange { o m m m } { }

```

(End definition for __siunitx_bookmark_cmd:Nn and others. These functions are documented on page ??.)

\c__siunitx_bookmark_seq Commands usable in bookmarks

```

173 \seq_const_from_clist:Nn \c__siunitx_bookmark_seq
174 { \qty , \num , \unit }

```

(End definition for \c__siunitx_bookmark_seq.)

Activate the document commands here: the unit macros are handled in siunitx-final.

```

175 \AtBeginDocument
176 {
177     \ifpackageloaded { hyperref }
178     {
179         \pdfstringdefDisableCommands
180         {
181             \seq_map_inline:Nn \c__siunitx_bookmark_seq

```

```

182         { \cs_set_eq:Nc #1 { \cs_to_str:N #1 ( pdfstring ~ context ) } }
183     }
184 }
185 { }
186 }

```

1.6 “Glue” commands

`__siunitx_angle:nnn` The document level interface for `\ang` needs some “glue” to work with the code-level API.

```

187 \cs_new_protected:Npn \__siunitx_angle:nnn #1#2#3
188 {
189     \tl_if_novalue:nTF {#2}
190     { \siunitx_angle:n {#1} }
191     {
192         \tl_if_novalue:nTF {#3}
193         { \siunitx_angle:nnn {#1} {#2} { } }
194         { \siunitx_angle:nnn {#1} {#2} {#3} }
195     }
196 }

```

(End definition for `__siunitx_angle:nnn`.)

1.7 Table column

User interfaces in tabular constructs are provided using the mechanisms from the `array` package.

```

197 \RequirePackage { array }

```

`__siunitx_declare_column:Nnn` Creating numerical columns requires that these are declared before anything else in `\NC@list`: this is necessary to work with optional arguments. This means a bit of manual effort after the simple declaration of a new column type. The token assigned to the column type is not fixed as this allows the same code to be used in compatibility with version 2.

```

198 \cs_new_protected:Npn \__siunitx_declare_column:Nnn #1#2#3
199 {
200     \newcolumntype {#1} { }
201     \cs_set_protected:Npn \__siunitx_tmp:w \NC@do ##1##2 \NC@do #1
202     { \NC@list { \NC@do ##1 \NC@do #1 ##2 } }
203     \exp_after:wN \__siunitx_tmp:w \the \NC@list
204     \exp_args:NNc \renewcommand * { \NC@rewrite@ #1 } [ 1 ] [ ]
205     {
206         \@temptokena \expandafter
207         {
208             \the \@temptokena
209             > {#2} c < {#3}
210         }
211         \NC@find
212     }
213 }

```

When `mdwtab` is loaded the syntax required is slightly different.

```

214 \AtBeginDocument
215 {
216   \@ifpackageloaded { mdwtab }
217   {
218     \cs_set_protected:Npn \__siunitx_declare_column:Nnn #1#2#3
219     {
220       \newcolumnntype {#1} [ 1 ] [ ]
221       { > {#2} c < {#3} }
222     }
223   }
224   { }
225 }
226 \AtBeginDocument
227 {
228   \__siunitx_declare_column:Nnn S
229   {
230     \keys_set:nn { siunitx } {#1}
231     \siunitx_cell_begin:w
232   }
233   { \siunitx_cell_end: }
234 }

```

(End definition for `__siunitx_declare_column:Nnn`.)

```

235 \endpackage

```

Part II

siunitx-angle – Formatting angles

1 Formatting angles

`\siunitx_angle:n`
`\siunitx_angle:nnn`

`\siunitx_angle:n {⟨angle⟩}`
`\siunitx_angle:nnn {⟨degrees⟩} {⟨minutes⟩} {⟨seconds⟩}`

Typeset the $\langle angle \rangle$ (which may be given as separate $\langle degree \rangle$, $\langle minute \rangle$ and $\langle second \rangle$ components). The $\langle angle \rangle$ (or components) may be given as expressions. The $\langle angle \rangle$ should be a number as understood by `\siunitx_format_number:nN`, with no uncertainty, exponent or imaginary part. The unit symbols for degrees, minutes and seconds are `\degree`, `\arcminute` and `\arcsecond`, respectively

1.1 Key-value options

The options defined by this submodule are available within the `l3keys siunitx` tree.

`angle-mode`

`angle-mode = ⟨choice⟩`

Selects how angles are formatted: a choice from the options `arc`, `decimal` and `input`. The option `arc` means that angles will always be typeset in arc (degree, minute, second) format, whilst `decimal` means that angles are typeset as a single decimal value. The `input` setting means that the input format (*i.e.* difference between `\siunitx_angle:n` and `\siunitx_angle:nnn`) is maintained. The standard setting is `input`.

`arc-separator`

`arc-separator = ⟨separator⟩`

Inserted between arc parts (degree, minute and second components). The standard setting is `\,`.

`arc-separator-over-decimal`

`arc-separator-over-decimal = true|false`

`fill-arc-degrees`

`fill-arc-degrees = true|false`

`fill-arc-minutes`

`fill-arc-minutes = true|false`

`fill-arc-seconds`

`fill-arc-seconds = true|false`

`number-angle-product`

`number-angle-product = ⟨separator⟩`

Inserted between the value of an angle and the unit (degree, minute or second component). The standard setting is `\,`.

2 siunitx-angle implementation

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 <@@=siunitx_angle>
```

Scratch space.

```
\l__siunitx_angle_tmp_bool
\l__siunitx_angle_tmp_dim
\l__siunitx_angle_tmp_tl
```

```
3 \bool_new:N \l__siunitx_angle_tmp_bool
4 \dim_new:N \l__siunitx_angle_tmp_dim
5 \tl_new:N \l__siunitx_angle_tmp_tl
```

(End definition for \l__siunitx_angle_tmp_bool, \l__siunitx_angle_tmp_dim, and \l__siunitx_angle_tmp_tl.)

```
\l__siunitx_angle_force_arc_bool
\l__siunitx_angle_force_decimal_bool
\l__siunitx_angle_astronomy_bool
\l__siunitx_angle_separator_tl
\l__siunitx_angle_fill_degrees_bool
\l__siunitx_angle_fill_minutes_bool
\l__siunitx_angle_fill_seconds_bool
\l__siunitx_angle_product_tl

6 \keys_define:nn { siunitx }
7 {
8   angle-mode .choice: ,
9   angle-mode / arc .code:n =
10  {
11    \bool_set_true:N \l__siunitx_angle_force_arc_bool
12    \bool_set_false:N \l__siunitx_angle_force_decimal_bool
13  } ,
14  angle-mode / decimal .code:n =
15  {
16    \bool_set_false:N \l__siunitx_angle_force_arc_bool
17    \bool_set_true:N \l__siunitx_angle_force_decimal_bool
18  } ,
19  angle-mode / input .code:n =
20  {
21    \bool_set_false:N \l__siunitx_angle_force_arc_bool
22    \bool_set_false:N \l__siunitx_angle_force_decimal_bool
23  } ,
24  angle-symbol-over-decimal .bool_set:N =
25    \l__siunitx_angle_astronomy_bool ,
26  arc-separator .tl_set:N =
27    \l__siunitx_angle_separator_tl ,
28  fill-arc-degrees .bool_set:N =
29    \l__siunitx_angle_fill_degrees_bool ,
30  fill-arc-minutes .bool_set:N =
31    \l__siunitx_angle_fill_minutes_bool ,
32  fill-arc-seconds .bool_set:N =
33    \l__siunitx_angle_fill_seconds_bool ,
34  number-angle-product .tl_set:N =
35    \l__siunitx_angle_product_tl
36  }
37 \bool_new:N \l__siunitx_angle_force_arc_bool
38 \bool_new:N \l__siunitx_angle_force_decimal_bool
```

(End definition for \l__siunitx_angle_force_arc_bool and others.)

`\siunitx_angle:n` The first step here is to force format conversion if required. Going to a decimal is easy, going to arc format is a bit more painful: avoid repeating calculations mainly for code readability.

```

39 \cs_new_protected:Npn \siunitx_angle:n #1
40 {
41   \bool_if:NTF \l__siunitx_angle_force_arc_bool
42     { \exp_args:Ne \__siunitx_angle_arc_convert:n { \fp_eval:n {#1} } }
43     {
44       \siunitx_number_parse:nN {#1} \l__siunitx_angle_degrees_tl
45       \tl_set:Nx \l__siunitx_angle_degrees_tl
46         { \siunitx_number_format:NN \l__siunitx_angle_degrees_tl \q_nil }
47       \__siunitx_angle_arc_print:VVV
48       \l__siunitx_angle_degrees_tl
49       \c_empty_tl
50       \c_empty_tl
51     }
52   }
53 \cs_new_protected:Npn \siunitx_angle:nnn #1#2#3
54 {
55   \bool_if:NTF \l__siunitx_angle_force_decimal_bool
56     {
57       \exp_args:Ne \siunitx_angle:n
58       { \fp_eval:n { #1 + (#2) / 60 + (#3) / 3600 } }
59     }
60     { \__siunitx_angle_arc_sign:nnn {#1} {#2} {#3} }
61   }
62 \cs_new_protected:Npn \__siunitx_angle_arc_convert:n #1
63 {
64   \use:x
65   {
66     \siunitx_angle:nnn
67     { \fp_eval:n { trunc(#1,0) } }
68     { \fp_eval:n { trunc((#1 - trunc(#1,0)) * 60,0) } }
69     {
70       \fp_eval:n
71       {
72         (
73           (#1 - trunc(#1,0)) * 60
74           - trunc((#1 - trunc(#1,0)) * 60,0)
75         )
76         * 60
77       }
78     }
79   }
80 }

```

(End definition for `\siunitx_angle:n`, `\siunitx_angle:nnn`, and `__siunitx_angle_arc_convert:n`. These functions are documented on page 8.)

```

\l__siunitx_angle_degrees_tl Space for formatting parsed numbers.
\l__siunitx_angle_minutes_tl 81 \tl_new:N \l__siunitx_angle_degrees_tl
\l__siunitx_angle_seconds_tl 82 \tl_new:N \l__siunitx_angle_minutes_tl
83 \tl_new:N \l__siunitx_angle_seconds_tl

```

(End definition for `\l__siunitx_angle_degrees_tl`, `\l__siunitx_angle_minutes_tl`, and `\l__siunitx_angle_seconds_tl`.)

`\l__siunitx_angle_sign_tl` For the “sign shuffle”.

84 `\tl_new:N \l__siunitx_angle_sign_tl`

(End definition for `\l__siunitx_angle_sign_tl`.)

`_siunitx_angle_arc_sign:nnn`

To get the sign in the right place whilst dealing with zero filling means doing some shuffling. That means doing processing of each number manually.

`_siunitx_angle_arc_sign:nn`

`_siunitx_angle_extract_sign:nnnnnnnn`

`_siunitx_angle_sign:nnnnnnnn`

85 `\cs_new_protected:Npn _siunitx_angle_arc_sign:nnn #1#2#3`

86 `{`

87 `\group_begin:`

88 `\keys_set:nn { siunitx }`

89 `{`

90 `input-close-uncertainty = ,`

91 `input-exponent-markers = ,`

92 `input-open-uncertainty = ,`

93 `input-uncertainty-signs =`

94 `}`

95 `\tl_clear:N \l__siunitx_angle_sign_tl`

96 `_siunitx_angle_arc_sign:nn {#1} { degrees }`

97 `_siunitx_angle_arc_sign:nn {#2} { minutes }`

98 `_siunitx_angle_arc_sign:nn {#3} { seconds }`

99 `\tl_if_empty:NF \l__siunitx_angle_sign_tl`

100 `{`

101 `\clist_map_inline:nn { degrees , minutes , seconds }`

102 `{`

103 `\tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }`

104 `{`

105 `\tl_set:cx { l__siunitx_angle_ ##1 _tl }`

106 `{`

107 `{ }`

108 `{ \exp_not:V \l__siunitx_angle_sign_tl }`

109 `\exp_after:wN \exp_after:wN \exp_after:wN`

110 `_siunitx_angle_sign:nnnnnnnn`

111 `\cs:w l__siunitx_angle_ ##1 _tl \cs_end:`

112 `}`

113 `\clist_map_break:`

114 `}`

115 `}`

116 `}`

117 `\clist_map_inline:nn { degrees , minutes , seconds }`

118 `{`

119 `\tl_if_empty:cF { l__siunitx_angle_ ##1 _tl }`

120 `{`

121 `\tl_set:cx { l__siunitx_angle_ ##1 _tl }`

122 `{`

123 `\exp_args:Nc \siunitx_number_format:NN`

124 `{ l__siunitx_angle_ ##1 _tl } \q_nil`

125 `}`

126 `}`

127 `}`

128 `_siunitx_angle_arc_print:VVV`

```

129         \l__siunitx_angle_degrees_tl
130         \l__siunitx_angle_minutes_tl
131         \l__siunitx_angle_seconds_tl
132     \group_end:
133 }
134 \cs_new_protected:Npn \__siunitx_angle_arc_sign:nn #1#2
135 {
136     \tl_if_blank:nTF {#1}
137     {
138         \bool_if:cTF { l__siunitx_angle_fill_ #2 _bool }
139         {
140             \tl_set:cn { l__siunitx_angle_ #2 _tl }
141             { { } { } { } { 0 } { } { } { } { } { 0 } }
142         }
143         { \tl_clear:c { l__siunitx_angle_ #2 _tl } }
144     }
145     {
146         \siunitx_number_parse:nN {#1} \l__siunitx_angle_tmp_tl
147         \exp_after:wN \__siunitx_angle_extract_sign:nnnnnnnn \l__siunitx_angle_tmp_tl {#2}
148     }
149 }
150 \cs_new_protected:Npn \__siunitx_angle_extract_sign:nnnnnnnn #1#2#3#4#5#6#7#8
151 {
152     \tl_if_blank:nTF {#2}
153     { \tl_set_eq:cN { l__siunitx_angle_ #8 _tl } \l__siunitx_angle_tmp_tl }
154     {
155         \tl_set:cn { l__siunitx_angle_ #8 _tl }
156         { {#1} { } { } {#3} {#4} {#5} {#6} {#7} }
157         \tl_set:Nn \l__siunitx_angle_sign_tl {#2}
158         \keys_set:nn { siunitx }
159         { input-comparators = , input-signs = }
160     }
161 }
162 \cs_new:Npn \__siunitx_angle_sign:nnnnnnnn #1#2#3#4#5#6#7
163 { \exp_not:n { {#3} {#4} {#5} {#6} {#7} } }

```

(End definition for __siunitx_angle_arc_sign:nnn and others.)

```

\l__siunitx_angle_marker_box For “astronomy style” angles.
\l__siunitx_angle_unit_box
164 \box_new:N \l__siunitx_angle_marker_box
165 \box_new:N \l__siunitx_angle_unit_box

```

(End definition for \l__siunitx_angle_marker_box and \l__siunitx_angle_unit_box.)

```

    \_siunitx_angle_arc_print:nnn The final stage of printing an angle is to put together the three parts: this works even for
    \_siunitx_angle_arc_print:VVV decimal angles as they will blank arguments for the other two parts The need to handle
    \_siunitx_angle_arc_print_auxi:nnn astronomy-style formatting means that the number has to be decomposed into parts.
    \_siunitx_angle_arc_print_auxii:w
    \_siunitx_angle_arc_print_auxiii:n
    \_siunitx_angle_arc_print_auxiv:NN
    \_siunitx_angle_arc_print_auxv:w
    \_siunitx_angle_arc_print_auxvi:n
166 \cs_new_protected:Npn \__siunitx_angle_arc_print:nnn #1#2#3
167 {
168     \__siunitx_angle_arc_print_auxi:nnn {#1} { \degree } {#2#3}
169     \__siunitx_angle_arc_print_auxi:nnn {#2} { \arcminute } {#3}
170     \__siunitx_angle_arc_print_auxi:nnn {#3} { \arcsecond } { }
171 }
172 \cs_generate_variant:Nn \__siunitx_angle_arc_print:nnn { VVV }

```



```

173 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxi:nnn #1#2#3
174 {
175   \tl_if_blank:nF {#1}
176   {
177     \bool_if:NTF \l__siunitx_angle_astronomy_bool
178     { \__siunitx_angle_arc_print_auxii:nw {#2} #1 \q_stop }
179     {
180       \__siunitx_angle_arc_print_auxv:w #1 \q_stop
181       \__siunitx_angle_arc_print_auxvi:n {#2}
182     }
183   \tl_if_blank:nF {#3}
184   {
185     \nobreak
186     \l__siunitx_angle_separator_tl
187   }
188 }
189 }
190 % \end{macrocode}
191 % To align the two parts of the astronomy-style marker, we need to allow
192 % for the |\scriptspace|.
193 % \begin{macrocode}
194 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxii:nw
195 #1#2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_stop
196 {
197   \mode_if_math:TF
198   { \bool_set_true:N \l__siunitx_angle_tmp_bool }
199   { \bool_set_false:N \l__siunitx_angle_tmp_bool }
200   \siunitx_print:nn { number } {#2#3}
201   \tl_if_blank:nTF {#5}
202   { \__siunitx_angle_arc_print_auxvi:n {#1} }
203   {
204     \hbox_set:Nn \l__siunitx_angle_marker_box
205     {
206       \__siunitx_angle_arc_print_auxiii:n
207       { \siunitx_print:nn { number } {#4} }
208     }
209     \hbox_set:Nn \l__siunitx_angle_unit_box
210     {
211       \__siunitx_angle_arc_print_auxiii:n
212       {
213         \siunitx_unit_format:nN {#1} \l__siunitx_angle_tmp_tl
214         \siunitx_print:nV { unit } \l__siunitx_angle_tmp_tl
215         \skip_horizontal:n { -\scriptspace }
216       }
217     }
218     \dim_compare:nNnTF { \box_wd:N \l__siunitx_angle_marker_box } >
219     { \box_wd:N \l__siunitx_angle_unit_box }
220     {
221       \__siunitx_angle_arc_print_auxiv:NN
222       \l__siunitx_angle_marker_box
223       \l__siunitx_angle_unit_box
224     }
225     {
226       \__siunitx_angle_arc_print_auxiv:NN

```

```

227         \l__siunitx_angle_unit_box
228         \l__siunitx_angle_marker_box
229     }
230     \hbox_set_to_wd:Nnn \l__siunitx_angle_marker_box
231     \l__siunitx_angle_tmp_dim
232     {
233         \hbox_overlap_right:n
234         { \box_use_drop:N \l__siunitx_angle_marker_box }
235         \hbox_overlap_right:n
236         { \box_use_drop:N \l__siunitx_angle_unit_box }
237         \tex_hfil:D
238     }
239     \box_use:N \l__siunitx_angle_marker_box
240     \skip_horizontal:N \scriptspace
241     \siunitx_print:nn { number } {#5}
242 }
243 }
244 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxiii:n #1
245 {
246     \bool_if:NTF \l__siunitx_angle_tmp_bool
247     { \ensuremath }
248     { \use:n }
249     {#1}
250 }
251 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxiv:NN #1#2
252 {
253     \dim_set:Nn \l__siunitx_angle_tmp_dim { \box_wd:N #1 }
254     \hbox_set_to_wd:Nnn #2
255     \l__siunitx_angle_tmp_dim
256     {
257         \tex_hss:D
258         \hbox_unpack_drop:N #2
259         \tex_hss:D
260     }
261 }
262 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxv:w
263 #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_stop
264 { \siunitx_print:nn { number } {#1#2#3#4#5} }
265 \cs_new_protected:Npn \__siunitx_angle_arc_print_auxvi:n #1
266 {
267     \nobreak
268     \l__siunitx_angle_product_tl
269     \siunitx_unit_format:nN {#1} \l__siunitx_angle_tmp_tl
270     \siunitx_print:nV { unit } \l__siunitx_angle_tmp_tl
271 }

```

(End definition for __siunitx_angle_arc_print:nnn and others.)

```

272 \keys_set:nn { siunitx }
273 {
274     angle-mode           = input ,
275     angle-symbol-over-decimal = false ,
276     arc-separator        =      ,
277     fill-arc-degrees     = false ,
278     fill-arc-minutes     = false ,

```

```
279     fill-arc-seconds      = false ,
280     number-angle-product  =
281   }
282 </package>
```

Part III

siunitx-quantity — Multi-part numbers

```
\siunitx_number_list:n \siunitx_number_list:n {\<entries>}
```

Prints the list of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced\ text \rangle$. Formatting of each number is carried out as described for `\siunitx_number_format:nN`.

```
\siunitx_number_product:n \siunitx_number_product:n {\<entries>}
```

Prints the series of numbers in the $\langle entries \rangle$, each of which should be given as a $\langle balanced\ text \rangle$. Formatting of each number is carried out as described for `\siunitx_number_format:nN`.

```
\siunitx_number_range:nn \siunitx_number_range:nn {\<start>} {\<end>}
```

Prints the range of numbers from the $\langle start \rangle$ to the $\langle end \rangle$, both of which are processed as described for `\siunitx_number_format:nN`.

Start the DocStrip guards.

```
1 \<package>
```

1 siunitx-compound implementation

1.1 Lists

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 \<@@=siunitx_list>
```

```
\l_siunitx_list_tmp_seq Scratch space.
```

```
\l_siunitx_list_tmp_tl 3 \seq_new:N \l_siunitx_list_tmp_seq
```

```
4 \tl_new:N \l_siunitx_list_tmp_tl
```

(End definition for `\l_siunitx_list_tmp_seq` and `\l_siunitx_list_tmp_tl`.)

```
\seq_use:NVVV
```

```
5 \cs_generate_variant:Nn \seq_use:Nnnn { NVVV }
```

(End definition for `\seq_use:NVVV`. This function is documented on page ??.)

```
\l_siunitx_list_separator_tl List options.
```

```
\l_siunitx_list_separator_final_tl 6 \bool_new:N \l_siunitx_list_bracket_bool
```

```
\l_siunitx_list_separator_pair_tl 7 \bool_new:N \l_siunitx_list_repeat_bool
```

```
\l_siunitx_list_bracket_bool 8 \keys_define:nn { siunitx }
```

```
9 {
```

```
10 list-final-separator .tl_set:N = \l_siunitx_list_separator_final_tl ,
```

```
11 list-pair-separator .tl_set:N = \l_siunitx_list_separator_pair_tl ,
```

```
12 list-separator .tl_set:N = \l_siunitx_list_separator_tl ,
```

```
13 list-mode .choice: ,
```

```

14 list-mode / bracket .code:n =
15 {
16   \bool_set_true:N \l__siunitx_list_bracket_bool
17   \bool_set_false:N \l__siunitx_list_repeat_bool
18 } ,
19 list-mode / repeat .code:n =
20 {
21   \bool_set_false:N \l__siunitx_list_bracket_bool
22   \bool_set_true:N \l__siunitx_list_repeat_bool
23 } ,
24 list-mode / single .code:n =
25 {
26   \bool_set_false:N \l__siunitx_list_bracket_bool
27   \bool_set_false:N \l__siunitx_list_repeat_bool
28 }
29 }

```

(End definition for `\l__siunitx_list_separator_tl` and others.)

`\siunitx_number_list:n` The hard work here can be done using a sequence: just a question of adding the data there.

```

30 \cs_new_protected:Npn \siunitx_number_list:n #1
31 {
32   \group_begin:
33   \seq_clear:N \l__siunitx_list_tmp_seq
34   \tl_map_inline:nn {#1}
35   {
36     \siunitx_number_format:nN {##1} \l__siunitx_list_tmp_tl
37     \seq_put_right:Nx \l__siunitx_list_tmp_seq
38     { \siunitx_print:nn { number } { \exp_not:V \l__siunitx_list_tmp_tl } }
39   }
40   \seq_use:NVVV \l__siunitx_list_tmp_seq
41   \l__siunitx_list_separator_pair_tl
42   \l__siunitx_list_separator_tl
43   \l__siunitx_list_separator_final_tl
44   \group_end:
45 }

```

(End definition for `\siunitx_number_list:n`. This function is documented on page 16.)

1.2 Products

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```

46 <@@=siunitx_product>

```

`\l__siunitx_product_tmp_seq` Scratch space.

```

\l__siunitx_product_tmp_tl
47 \seq_new:N \l__siunitx_product_tmp_seq
48 \tl_new:N \l__siunitx_product_tmp_tl

```

(End definition for `\l__siunitx_product_tmp_seq` and `\l__siunitx_product_tmp_tl`.)

\l__siunitx_product_mode_tl Options for products.

```

\l__siunitx_product_phrase_tl 49 \tl_new:N \l__siunitx_product_mode_tl
\l__siunitx_product_symbol_tl 50 \keys_define:nn { siunitx }
51 {
52   product-mode .choices:nn =
53     { phrase , symbol }
54     { \tl_set_eq:NN \l__siunitx_product_mode_tl \l_keys_choice_tl } ,
55   product-phrase .tl_set:N = \l__siunitx_product_phrase_tl ,
56   product-symbol .tl_set:N = \l__siunitx_product_symbol_tl
57 }

```

(End definition for \l__siunitx_product_mode_tl, \l__siunitx_product_phrase_tl, and \l__siunitx-product_symbol_tl.)

\seq_use:Nx

```

58 \cs_generate_variant:Nn \seq_use:Nn { Nx }

```

(End definition for \seq_use:Nx. This function is documented on page ??.)

\siunitx_number_product:n Much like lists, but with two possible output methods and a simpler loop.

```

\__siunitx_product_phrase: 59 \cs_new_protected:Npn \siunitx_number_product:n #1
\__siunitx_product_symbol: 60 {
61   \group_begin:
62     \seq_clear:N \l__siunitx_product_tmp_seq
63     \tl_map_inline:nn {#1}
64     {
65       \siunitx_number_format:nN {##1} \l__siunitx_product_tmp_tl
66       \seq_put_right:Nx \l__siunitx_product_tmp_seq
67       { \siunitx_print:nn { number } { \exp_not:V \l__siunitx_product_tmp_tl } }
68     }
69     \seq_use:Nx \l__siunitx_product_tmp_seq
70     { \exp_not:c { __siunitx_product_ \l__siunitx_product_mode_tl : } }
71   \group_end:
72 }
73 \cs_new_protected:Npn \__siunitx_product_phrase:
74 { \tl_use:N \l__siunitx_product_phrase_tl }
75 \cs_new_protected:Npn \__siunitx_product_symbol:
76 {
77   \tl_set:Nx \l__siunitx_product_tmp_tl
78   { { } \exp_not:N \l__siunitx_product_symbol_tl { } }
79   \siunitx_print:nV { number } \l__siunitx_product_tmp_tl
80 }

```

(End definition for \siunitx_number_product:n, __siunitx_product_phrase:, and __siunitx-product_symbol:. This function is documented on page 16.)

1.3 Ranges

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```

81 <@@=siunitx_range>

```

\l__siunitx_range_tmp_tl Scratch space.

```

82 \tl_new:N \l__siunitx_range_tmp_tl

```

(End definition for \l__siunitx_range_tmp_tl.)

```

\l__siunitx_range_phrase_tl
\l__siunitx_range_bracket_bool
\l__siunitx_range_repeat_bool
83 \bool_new:N \l__siunitx_range_bracket_bool
84 \bool_new:N \l__siunitx_range_repeat_bool
85 \keys_define:nn { siunitx }
86 {
87   range-phrase .tl_set:N = \l__siunitx_range_phrase_tl ,
88   range-mode .choice: ,
89   range-mode / bracket .code:n =
90   {
91     \bool_set_true:N \l__siunitx_range_bracket_bool
92     \bool_set_false:N \l__siunitx_range_repeat_bool
93   } ,
94   range-mode / repeat .code:n =
95   {
96     \bool_set_false:N \l__siunitx_range_bracket_bool
97     \bool_set_true:N \l__siunitx_range_repeat_bool
98   } ,
99   range-mode / single .code:n =
100   {
101     \bool_set_false:N \l__siunitx_range_bracket_bool
102     \bool_set_false:N \l__siunitx_range_repeat_bool
103   }
104 }

```

(End definition for \l__siunitx_range_phrase_tl, \l__siunitx_range_bracket_bool, and \l__siunitx_range_repeat_bool.)

\siunitx_number_range:nn A range of numbers is quite straight-forward.

```

105 \cs_new_protected:Npn \siunitx_number_range:nn #1#2
106 {
107   \siunitx_number_format:nN {#1} \l__siunitx_range_tmp_tl
108   \siunitx_print:nV { number } \l__siunitx_range_tmp_tl
109   \tl_use:N \l__siunitx_range_phrase_tl
110   \siunitx_number_format:nN {#2} \l__siunitx_range_tmp_tl
111   \siunitx_print:nV { number } \l__siunitx_range_tmp_tl
112 }

```

(End definition for \siunitx_number_range:nn. This function is documented on page 16.)

1.4 Standard settings for module options

Some of these follow naturally from the point of definition (e.g. boolean variables are always false to begin with), but for clarity everything is set here.

```

113 \keys_set:nn { siunitx }
114 {
115   list-final-separator = \text { ~ and ~ } ,
116   list-pair-separator  = \text { ~ and ~ } ,
117   list-separator       = \text { , ~ } ,
118   list-mode             = repeat ,
119   product-mode          = symbol ,
120   product-phrase       = \text { ~ by ~ } ,
121   product-symbol       = \times ,

```

```
122     range-phrase      = \text { ~ to ~ } ,
123     range-mode        = repeat
124   }
125 \end{package}
```


Part IV

siunitx-number – Parsing and formatting numbers

1 Formatting numbers

<code>\siunitx_number_format:nN</code>	<code>\siunitx_number_format:nN {<number>} <tl var></code>
<code>\siunitx_number_format:VN</code>	

<code>\siunitx_number_parse:nN</code>	<code>\siunitx_number_parse:nN {<number>} <tl var></code>
---------------------------------------	---

Parses the *number* and stores the resulting internal representation in the *<tl var>*. The parsing is influenced by the various key–value settings for numerical input. The *<number>* should comprise a single real value, possibly with comparator, uncertainty and exponent parts. If the number is invalid, or if number parsing is disabled, the result will be an entirely empty *<tl var>*.

The structure of a valid number is:

$$\{ \langle \textit{comparator} \rangle \} \{ \langle \textit{sign} \rangle \} \{ \langle \textit{integer} \rangle \} \{ \langle \textit{decimal} \rangle \} \{ \langle \textit{uncertainty} \rangle \} \{ \langle \textit{exponent sign} \rangle \} \{ \langle \textit{exponent} \rangle \}$$

where the two sign parts must be single tokens if present, and all other components must be given in braces. The number will have at least one digit for both the *<integer>* and *<exponent>* parts: these are required. The *<uncertainty>* part should either be blank or contain an *<identifier>* (as a brace group), followed by one or more data entries. Valid *<identifiers>* currently are

S A single symmetrical uncertainty (*e.g.* a statistical standard uncertainty)

<code>\siunitx_number_process:NN</code>	<code>\siunitx_number_process:N <tl var1> <tl var2></code>
---	--

Applies a set of number processing operations to the *<internal number>* stored in the *<tl var1>*, *viz.* in order

1. Dropping uncertainty
2. Converting to scientific mode (or similar)
3. Rounding
4. Dropping zero decimal part
5. Forcing a minimum number of digits

with the result stored in *<tl var2>*.

<code>\siunitx_number_format:N</code>	☆	<code>\siunitx_number_format:N</code>	$\langle number \rangle$
<code>\siunitx_number_format:NN</code>	☆	<code>\siunitx_number_format:NN</code>	$\langle number \rangle$ $\langle marker \rangle$

Formats the $\langle number \rangle$ (in the `siunitx` internal format), producing the result in a form suitable for typesetting in math mode. The details for the formatting are controlled by a number of key–value options. Note that *formatting* does not apply any manipulation (processing) to the number. This function is usable in an `e-` or `x-` type expansion, and further uncontrolled expansion is prevented by appropriate use of `\exp_not:n` internally.

In the `NN` version, the $\langle marker \rangle$ token is inserted at each possible alignment position in the output, *viz.*

- Between the comparator and the integer (*before* any sign for the integer)
- Both sides of the decimal marker
- Both sides of the separated uncertainty sign (*i.e.* after the decimal part and before any integer uncertainty part)
- Both sides of the decimal marker for a separated uncertainty
- Both sides of the multiplication symbol for the exponent part.

<code>\siunitx_if_number_p:n</code>	★	<code>\siunitx_if_number_token:NTF</code>	$\{\langle tokens \rangle\}$
<code>\siunitx_if_number:nTF</code>	★	$\{\langle true\ code \rangle\}$	$\{\langle false\ code \rangle\}$

Determines if the $\langle tokens \rangle$ form a valid number which can be fully parsed by `siunitx`.

<code>\siunitx_if_number_token:NTF</code>	$\{\langle token \rangle\}$
	$\{\langle true\ code \rangle\}$ $\{\langle false\ code \rangle\}$

Determines if the $\langle token \rangle$ is valid in a number based on those tokens currently set up for detection in a number.

`\l_siunitx_number_parse_bool`

A switch to control whether any parsing is attempted for numbers.

`\l_siunitx_number_input_decimal_tl`
`\l_siunitx_number_output_decimal_tl`

The list of possible input decimal marker(s), and the output marker.

1.1 Key–value options

The options defined by this submodule are available within the `l3keys siunitx` tree.

`bracket-negative-numbers` `bracket-negative-numbers = true|false`

`drop-exponent` `drop-exponent = true|false`

`drop-uncertainty` `drop-uncertainty = true|false`

<u>drop-zero-decimal</u>	drop-zero-decimal = true false
<u>evaluate-expression</u>	evaluate-expression = true false
<u>exponent-base</u>	exponent-base = $\langle base \rangle$
<u>exponent-mode</u>	exponent-mode = engineering fixed input scientific
<u>exponent-product</u>	exponent-product = $\langle symbol \rangle$
<u>expression</u>	expression = $\langle expression \rangle$
<u>fixed-exponent</u>	fixed-exponent = $\langle exponent \rangle$
<u>group-digits</u>	group-digits = all decimal integer none
<u>group-minimum-digits</u>	group-minimum-digits = $\langle value \rangle$
<u>group-separator</u>	group-separator = $\langle symbol \rangle$
<u>input-close-uncertainty</u>	input-close-uncertainty = $\langle tokens \rangle$
<u>input-comparators</u>	input-comparators = $\langle tokens \rangle$
<u>input-close-uncertainty</u>	input-close-uncertainty = $\langle tokens \rangle$
<u>input-decimal-markers</u>	input-decimal-markers = $\langle tokens \rangle$
<u>input-digits</u>	input-digits = $\langle tokens \rangle$
<u>input-exponent-markers</u>	input-exponent-markers = $\langle tokens \rangle$
<u>input-open-uncertainty</u>	input-open-uncertainty = $\langle tokens \rangle$
<u>input-signs</u>	input-signs = $\langle tokens \rangle$

<u>input-uncertainty-signs</u>	input-uncertainty-signs = $\langle tokens \rangle$
<u>minimum-decimal-digits</u>	minimum-decimal-digits = $\langle min \rangle$
<u>minimum-integer-digits</u>	minimum-integer-digits = $\langle min \rangle$
<u>negative-color</u>	negative-color = $\langle color \rangle$
<u>number-close-bracket</u>	number-close-bracket = $\langle symbol \rangle$
<u>number-open-bracket</u>	number-open-bracket = $\langle symbol \rangle$
<u>output-close-uncertainty</u>	output-close-uncertainty = $\langle symbol \rangle$
<u>output-decimal-marker</u>	output-decimal-marker = $\langle symbol \rangle$
<u>output-open-uncertainty</u>	output-open-uncertainty = $\langle symbol \rangle$
<u>parse-numbers</u>	parse-numbers = true false
<u>print-implicit-plus</u>	print-implicit-plus = true false
<u>print-unity-mantissa</u>	print-unity-mantissa = true false
<u>print-zero-exponent</u>	print-zero-exponent = true false
<u>round-half</u>	round-half = even up
<u>round-minimum</u>	round-minimum = $\langle min \rangle$
<u>round-mode</u>	round-mode = figures none places uncertainty
<u>round-pad</u>	round-pad = true false
<u>round-precision</u>	round-precision = $\langle precision \rangle$

<hr/> <hr/> separate-uncertainty	separate-uncertainty = true false
<hr/> <hr/> track-explicit-plus	track-explicit-plus = true false
<hr/> <hr/> uncertainty-separator	uncertainty-separator = $\langle separator \rangle$
<hr/> <hr/> tight-spacing	tight-spacing = true false

2 siunitx-number implementation

Start the DocStrip guards.

```
1  $\langle *package \rangle$ 
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2  $\langle @@=siunitx\_number \rangle$ 
```

2.1 Initial set-up

Variants not provided by expl3.

```
3 \cs_generate_variant:Nn \tl_if_blank:nTF { f }
4 \cs_generate_variant:Nn \tl_if_blank_p:n { f }
5 \cs_generate_variant:Nn \tl_if_in:NnTF { NV }
6 \cs_generate_variant:Nn \tl_replace_all:Nnn { NnV }
```

```
\l__siunitx_number_tmp_tl Scratch space.
```

```
7 \tl_new:N \l__siunitx_number_tmp_tl
```

(End definition for `\l__siunitx_number_tmp_tl`.)

2.2 Main formatting routine

```
\l_siunitx_number_formatted_tl A token list for the final formatted result: may or may not be generated by the parser, depending on settings which are active.
```

```
8 \tl_new:N \l_siunitx_number_formatted_tl
```

(End definition for `\l_siunitx_number_formatted_tl`.)

```
\l_siunitx_number_parse_bool Tracks whether to parse numbers: public as this may affect other behaviors.
```

```
9 \tl_new:N \l_siunitx_number_parse_bool
```

(End definition for `\l_siunitx_number_parse_bool`. This variable is documented on page 22.)

```
\l_siunitx_number_parse_bool Top-level options.
```

```
10 \keys_define:nn { siunitx }
11 {
12   parse-numbers .bool_set:N = \l_siunitx_number_parse_bool
13 }
```

(End definition for `\l_siunitx_number_parse_bool`. This variable is documented on page 22.)

```

\siunitx_number_format:nN
\siunitx_number_format:VN
\__siunitx_number_format:nN
14 \cs_new_protected:Npn \siunitx_number_format:nN #1#2
15 {
16   \group_begin:
17   \bool_if:NTF \l_siunitx_number_parse_bool
18   {
19     \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
20     \siunitx_number_process:NN \l__siunitx_number_parsed_tl \l__siunitx_number_parsed_tl
21     \tl_set:Nx \l__siunitx_number_formatted_tl
22     { \siunitx_number_format:N \l__siunitx_number_parsed_tl }
23   }
24   { \tl_set:Nn \l__siunitx_number_formatted_tl { \ensuremath {#1} } }
25   \exp_args:NNNV \group_end:
26   \tl_set:Nn #2 \l__siunitx_number_formatted_tl
27 }
28 \cs_generate_variant:Nn \siunitx_number_format:nN { V }

```

(End definition for `\siunitx_number_format:nN` and `__siunitx_number_format:nN`. This function is documented on page 21.)

2.3 Parsing numbers

Before numbers can be manipulated or formatted they need to be parsed into an internal form. In particular, if multiple code paths are to be avoided, it is necessary to do such parsing even for relatively simple cases such as converting `1e10` to `1 \times 10^{\{10\}}`.

Storing the result of such parsing can be done in a number of ways. In the first version of `siunitx` a series of separate data stores were used. This is potentially quite fast (as recovery of items relies only on `TeX`'s hash table) but makes managing the various data entries somewhat tedious and error-prone. For version two of the package, a single data structure (property list) was used for each part of the parsed number. Whilst this is easy to manage and extend, it is somewhat slower as at a `TeX` level there are repeated pack-unpack steps. In particular, the fact that there are a limited number of items to track for a “number” means that a more efficient approach is desirable (contrast parsing units, which is open-ended and therefore fits well with using a property list).

In this release, the structure of a valid number is:

$$\{\langle comparator \rangle\} \langle sign \rangle \{\langle integer \rangle\} \{\langle decimal \rangle\} \{\langle uncertainty \rangle\} \langle exponent sign \rangle \{\langle exponent \rangle\}$$

where the two sign parts must be single tokens and all other components must be given in braces. *All* of the components must be present in a stored number (*i.e.* at the end of parsing). The number must have at least one digit for both the `\langle integer \rangle` and `\langle exponent \rangle` parts.

`\l_siunitx_number_input_decimal_tl` The input decimal markers(s).

```

29 \tl_new:N \l_siunitx_number_input_decimal_tl

```

(End definition for `\l_siunitx_number_input_decimal_tl`. This variable is documented on page 22.)

```

\l_siunitx_number_expression_bool
\l_siunitx_number_input_uncert_close_tl
\l_siunitx_number_input_comparator_tl
\l_siunitx_number_input_digit_tl
\l_siunitx_number_input_exponent_tl
\l_siunitx_number_input_ignore_tl
\l_siunitx_number_input_uncert_open_tl
\l_siunitx_number_input_sign_tl
\l_siunitx_number_input_uncert_sign_tl
\l_siunitx_number_explicit_plus_bool
\l_siunitx_number_expression:n

```

Options which determine the various valid parts of a parsed number.

```

30 \keys_define:nn { siunitx }
31 {
32   evaluate-expression .bool_set:N =
33     \l_siunitx_number_expression_bool ,
34   expression .code:n =
35     \cs_set:Npn \__siunitx_number_expression:n ##1 {#1} ,
36   input-close-uncertainty .tl_set:N =
37     \l_siunitx_number_input_uncert_close_tl ,
38   input-comparators .tl_set:N =
39     \l_siunitx_number_input_comparator_tl ,
40   input-decimal-markers .tl_set:N =
41     \l_siunitx_number_input_decimal_tl ,
42   input-digits .tl_set:N =
43     \l_siunitx_number_input_digit_tl ,
44   input-exponent-markers .tl_set:N =
45     \l_siunitx_number_input_exponent_tl ,
46   input-ignore .tl_set:N =
47     \l_siunitx_number_input_ignore_tl ,
48   input-open-uncertainty .tl_set:N =
49     \l_siunitx_number_input_uncert_open_tl ,
50   input-signs .tl_set:N =
51     \l_siunitx_number_input_sign_tl ,
52   input-uncertainty-signs .code:n =
53     {
54       \tl_set:Nn \l_siunitx_number_input_uncert_sign_tl {#1}
55       \tl_map_inline:nn {#1}
56       {
57         \tl_if_in:NnF \l_siunitx_number_input_sign_tl {##1}
58         { \tl_put_right:Nn \l_siunitx_number_input_sign_tl {##1} }
59       }
60     } ,
61   parse-numbers .bool_set:N =
62     \l_siunitx_number_parse_bool ,
63   track-explicit-plus .bool_set:N =
64     \l_siunitx_number_explicit_plus_bool
65   }
66 \cs_new:Npn \__siunitx_number_expression:n #1 { }
67 \tl_new:N \l_siunitx_number_input_uncert_sign_tl

```

(End definition for \l_siunitx_number_expression_bool and others.)

```
\l_siunitx_number_arg_tl
```

The input argument or a part thereof, depending on the position in the parsing routine.

```
68 \tl_new:N \l_siunitx_number_arg_tl
```

(End definition for \l_siunitx_number_arg_tl.)

```
\l_siunitx_number_comparator_tl
```

A comparator, if found, is held here.

```
69 \tl_new:N \l_siunitx_number_comparator_tl
```

(End definition for \l_siunitx_number_comparator_tl.)

```
\l_siunitx_number_exponent_tl
```

The exponent part of a parsed number. It is easiest to find this relatively early in the parsing process, but as it needs to go at the end of the internal format is held separately until required.

```
70 \tl_new:N \l_siunitx_number_exponent_tl
```

(End definition for \l__siunitx_number_exponent_tl.)

\l__siunitx_number_flex_tl In a number with an uncertainty, the exact meaning of a second part is not fully resolved until parsing is complete. That is handled using this “flexible” store.

71 \tl_new:N \l__siunitx_number_flex_tl

(End definition for \l__siunitx_number_flex_tl.)

\l__siunitx_number_parsed_tl The number parsed into internal format.

72 \tl_new:N \l__siunitx_number_parsed_tl

(End definition for \l__siunitx_number_parsed_tl.)

\l__siunitx_number_input_tl The numerical input exactly as given by the user.

73 \tl_new:N \l__siunitx_number_input_tl

(End definition for \l__siunitx_number_input_tl.)

\l__siunitx_number_partial_tl To avoid needing to worry about the fact that the final data stores are somewhat tricky to add to token-by-token, a simple store is used to build up the parsed part of a number before transferring in one go.

74 \tl_new:N \l__siunitx_number_partial_tl

(End definition for \l__siunitx_number_partial_tl.)

\l__siunitx_number_validate_bool Used to set up for validation with no error production.

75 \bool_new:N \l__siunitx_number_validate_bool

(End definition for \l__siunitx_number_validate_bool.)

\siunitx_number_parse:nN After some initial set up, the parser expands the input and then replaces as far as possible tricky tokens with ones that can be handled using delimited arguments. To avoid multiple conditionals here, the parser is set up as a chain of commands initially, with a loop only later. This avoids more conditionals than are necessary.

76 \cs_new_protected:Npn \siunitx_number_parse:nN #1#2

77 {

78 \bool_if:NTF \l__siunitx_number_parse_bool

79 { __siunitx_number_parse:nN {#1} #2 }

80 { \tl_clear:N #2 }

81 }

82 \cs_new_protected:Npn __siunitx_number_parse:nN #1#2

83 {

84 \group_begin:

85 \tl_clear:N \l__siunitx_number_parsed_tl

86 \protected@edef \l__siunitx_number_arg_tl

87 {

88 \bool_if:NTF \l__siunitx_number_expression_bool

89 { \fp_eval:n { __siunitx_number_expression:n {#1} } }

90 {#1}

91 }

92 \tl_set_eq:NN \l__siunitx_number_input_tl \l__siunitx_number_arg_tl

93 __siunitx_number_parse_replace:

94 \tl_if_empty:NF \l__siunitx_number_arg_tl

95 { __siunitx_number_parse_comparator: }

96 __siunitx_number_parse_check:


```

97   \exp_args:NNNV \group_end:
98   \tl_set:Nn #2 \l__siunitx_number_parsed_tl
99 }

```

(End definition for `\siunitx_number_parse:nN` and `_siunitx_number_parse:nN`. This function is documented on page 21.)

`_siunitx_number_parse_check:` After the loop there is one case that might need tidying up. If a separated uncertainty was found it will be currently in `\l__siunitx_number_flex_tl` and needs moving. A series of tests pick up that case, then the check is made that some content was found

```

100 \cs_new_protected:Npn \_siunitx_number_parse_check:
101 {
102   \tl_if_empty:NF \l__siunitx_number_flex_tl
103   {
104     \bool_lazy_and:nnTF
105     {
106       \tl_if_blank_p:f
107       { \exp_after:wN \use_iv:nnnn \l__siunitx_number_parsed_tl }
108     }
109     {
110       \tl_if_blank_p:f
111       { \exp_after:wN \use_iv:nnnn \l__siunitx_number_flex_tl }
112     }
113     {
114       \tl_set:Nx \l__siunitx_number_tmp_tl
115       { \exp_after:wN \use_i:nnnn \l__siunitx_number_flex_tl }
116       \tl_if_in:NVTF \l__siunitx_number_input_uncert_sign_tl
117       \l__siunitx_number_tmp_tl
118       { \_siunitx_number_parse_combine_uncert: }
119       { \tl_clear:N \l__siunitx_number_parsed_tl }
120     }
121     { \tl_clear:N \l__siunitx_number_parsed_tl }
122   }
123   \tl_if_empty:NTF \l__siunitx_number_parsed_tl
124   {
125     \bool_if:NF \l__siunitx_number_validate_bool
126     {
127       \msg_error:nnx { siunitx } { invalid-number }
128       { \exp_not:V \l__siunitx_number_input_tl }
129     }
130   }
131   { \_siunitx_number_parse_finalise: }
132 }

```

(End definition for `_siunitx_number_parse_check:.`)

`_siunitx_number_parse_combine_uncert:` Conversion of a second numerical part to an uncertainty needs a bit of work. The first step is to extract the useful information from the two stores: the sign, integer and decimal parts from the real number and the integer and decimal parts from the second number. That is done using the input stack to avoid lots of assignments.

```

133 \cs_new_protected:Npn \_siunitx_number_parse_combine_uncert:
134 {
135   \exp_after:wN \exp_after:wN \exp_after:wN
136   \_siunitx_number_parse_combine_uncert_auxi:nnnnnnnn

```

```

137     \exp_after:wN \l__siunitx_number_parsed_tl \l__siunitx_number_flex_tl
138   }

```

Here, #4, #5 and #8 are all junk arguments simply there to mop up tokens, while #1 will be recovered later from \l__siunitx_number_parsed_tl so does not need to be passed about. The difference in places between the two decimal parts is now found: this is done just once to avoid having to parse token lists twice. The value is then used to generate a number of filler 0 tokens, and these are added to the appropriate part of the number. Finally, everything is recombined: the integer part only needs a test to avoid an empty main number.

```

139 \cs_new_protected:Npn
140   \__siunitx_number_parse_combine_uncert_auxi:nnnnnnn #1#2#3#4#5#6#7#8
141   {
142     \int_compare:nNnTF { \tl_count:n {#6} } > { \tl_count:n {#2} }
143     {
144       \tl_clear:N \l__siunitx_number_parsed_tl
145       \tl_clear:N \l__siunitx_number_flex_tl
146     }
147     {
148       \__siunitx_number_parse_combine_uncert_auxii:fnnnn
149       { \int_eval:n { \tl_count:n {#3} - \tl_count:n {#7} } }
150       {#2} {#3} {#6} {#7}
151     }
152   }
153 \cs_new_protected:Npn
154   \__siunitx_number_parse_combine_uncert_auxii:nnnnn #1
155   {
156     \__siunitx_number_parse_combine_uncert_auxiii:fnnnnn
157     { \prg_replicate:nn { \int_abs:n {#1} } { 0 } }
158     {#1}
159   }
160 \cs_generate_variant:Nn \__siunitx_number_parse_combine_uncert_auxii:nnnnn { f }
161 \cs_new_protected:Npn
162   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn #1#2#3#4#5#6
163   {
164     \int_compare:nNnTF {#2} > 0
165     {
166       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
167       {#3} {#4} {#5} { #6 #1 }
168     }
169     {
170       \__siunitx_number_parse_combine_uncert_auxiv:nnnn
171       {#3} { #4 #1 } {#5} {#6}
172     }
173   }
174 \cs_generate_variant:Nn
175   \__siunitx_number_parse_combine_uncert_auxiii:nnnnnn { f }
176 \cs_new_protected:Npn
177   \__siunitx_number_parse_combine_uncert_auxiv:nnnn #1#2#3#4
178   {
179     \tl_set:Nx \l__siunitx_number_parsed_tl
180     {
181       { \tl_head:V \l__siunitx_number_parsed_tl }
182       { \exp_not:n {#1} }

```

```

183     {
184         \bool_lazy_and:nnTF
185         { \tl_if_blank_p:n {#2} }
186         { ! \tl_if_blank_p:n {#4} }
187         { 0 }
188         { \exp_not:n {#2} }
189     }
190     {
191         \__siunitx_number_parse_combine_uncert_auxv:w #3#4
192         \q_recursion_tail \q_recursion_stop
193     }
194 }
195 }

```

A short routine to remove any leading zeros in the uncertainty part, which are not needed for the compact representation used by the module.

```

196 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxv:w #1
197 {
198     \quark_if_recursion_tail_stop:N #1
199     \str_if_eq:nnTF {#1} { 0 }
200     { \__siunitx_number_parse_combine_uncert_auxv:w }
201     { \__siunitx_number_parse_combine_uncert_auxvi:w #1 }
202 }
203 \cs_new:Npn \__siunitx_number_parse_combine_uncert_auxvi:w
204 #1 \q_recursion_tail \q_recursion_stop
205 {
206     \tl_if_blank:nF {#1}
207     { { S } { \exp_not:n {#1} } }
208 }

```

(End definition for `__siunitx_number_parse_combine_uncert:` and others.)

`__siunitx_number_parse_comparator:`
`__siunitx_number_parse_comparator_aux:Nw`

A comparator has to be the very first token in the input. A such, the test for this can be very fast: grab the first token, do a check and if appropriate store the result.

```

209 \cs_new_protected:Npn \__siunitx_number_parse_comparator:
210 {
211     \exp_after:wN \__siunitx_number_parse_comparator_aux:Nw
212     \l__siunitx_number_arg_tl \q_stop
213 }
214 \cs_new_protected:Npn \__siunitx_number_parse_comparator_aux:Nw #1#2 \q_stop
215 {
216     \tl_if_in:NnTF \l__siunitx_number_input_comparator_tl {#1}
217     {
218         \tl_set:Nn \l__siunitx_number_comparator_tl {#1}
219         \tl_set:Nn \l__siunitx_number_arg_tl {#2}
220     }
221     { \tl_clear:N \l__siunitx_number_comparator_tl }
222     \tl_if_empty:NF \l__siunitx_number_arg_tl
223     { \__siunitx_number_parse_sign: }
224 }

```

(End definition for `__siunitx_number_parse_comparator:` and `__siunitx_number_parse_comparator_aux:Nw`.)

`_siunitx_number_parse_exponent:` An exponent part of a number has to come at the end and can only occur once. Thus it
`_siunitx_number_parse_exponent_auxi:w` is relatively easy to parse. First, there is a check that an exponent part is allowed, and
`_siunitx_number_parse_exponent_auxii:nn` if so a split is made (the previous part of the chain checks that there is some content in
`_siunitx_number_parse_exponent_auxiii:Nw` `\l__siunitx_number_arg_tl` before calling this function). After splitting, if there is no
`_siunitx_number_parse_exponent_auxiv:nn` exponent then simply save a default. Otherwise, check for a sign and then store either
`_siunitx_number_parse_exponent_zero_test:N` this or an implicit plus, and the digits after a check that nothing else is present after
`_siunitx_number_parse_exponent_check:N` the **e**. The only slight complication to all of this is allowing an arbitrary token in the
`_siunitx_number_parse_exponent_cleanup:N` input to represent the exponent: this is done by setting any exponent tokens to the first
of the allowed list, then using that in a delimited argument set up. Once an exponent
part is found, there is a loop to check that each of the tokens is a digit then a tidy up
step to remove any leading zeros.

```

225 \cs_new_protected:Npn \_siunitx_number_parse_exponent:
226 {
227   \tl_if_empty:NTF \l__siunitx_number_input_exponent_tl
228   {
229     \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 }
230     \tl_if_empty:NF \l__siunitx_number_parsed_tl
231     { \_siunitx_number_parse_loop: }
232   }
233   {
234     \tl_set:Nx \l__siunitx_number_tmp_tl
235     { \tl_head:V \l__siunitx_number_input_exponent_tl }
236     \tl_map_inline:Nn \l__siunitx_number_input_exponent_tl
237     {
238       \tl_replace_all:NnV \l__siunitx_number_arg_tl
239       {##1} \l__siunitx_number_tmp_tl
240     }
241     \use:x
242     {
243       \cs_set_protected:Npn
244       \exp_not:N \_siunitx_number_parse_exponent_auxi:w
245       ####1 \exp_not:V \l__siunitx_number_tmp_tl
246       ####2 \exp_not:V \l__siunitx_number_tmp_tl
247       ####3 \exp_not:N \q_stop
248     }
249     { \_siunitx_number_parse_exponent_auxii:nn {##1} {##2} }
250     \use:x
251     {
252       \_siunitx_number_parse_exponent_auxi:w
253       \exp_not:V \l__siunitx_number_arg_tl
254       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_nil
255       \exp_not:V \l__siunitx_number_tmp_tl \exp_not:N \q_stop
256     }
257   }
258 }
259 \cs_new_protected:Npn \_siunitx_number_parse_exponent_auxi:w { }
260 \cs_new_protected:Npn \_siunitx_number_parse_exponent_auxii:nn #1#2
261 {
262   \quark_if_nil:nTF {#2}
263   { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
264   {
265     \tl_set:Nn \l__siunitx_number_arg_tl {#1}
266     \tl_if_blank:nTF {#2}

```

```

267         { \tl_clear:N \l__siunitx_number_parsed_tl }
268         { \__siunitx_number_parse_exponent_auxiii:Nw #2 \q_stop }
269     }
270     \tl_if_empty:NF \l__siunitx_number_parsed_tl
271     { \__siunitx_number_parse_loop: }
272 }
273 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiii:Nw #1#2 \q_stop
274 {
275     \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
276     { \__siunitx_number_parse_exponent_auxiv:nn {#1} {#2} }
277     { \__siunitx_number_parse_exponent_auxiv:nn { } {#1#2} }
278     \tl_if_empty:NT \l__siunitx_number_exponent_tl
279     { \tl_clear:N \l__siunitx_number_parsed_tl }
280 }
281 \cs_new_protected:Npn \__siunitx_number_parse_exponent_auxiv:nn #1#2
282 {
283     \bool_lazy_or:nnTF
284     { \l__siunitx_number_explicit_plus_bool }
285     { ! \str_if_eq_p:nn {#1} { + } }
286     { \tl_set:Nn \l__siunitx_number_exponent_tl { {#1} } }
287     { \tl_set:Nn \l__siunitx_number_exponent_tl { { } } }
288     \tl_if_blank:nTF {#2}
289     { \tl_clear:N \l__siunitx_number_parsed_tl }
290     {
291         \__siunitx_number_parse_exponent_zero_test:N #2
292         \q_recursion_tail \q_recursion_stop
293     }
294 }
295 \cs_new_protected:Npn \__siunitx_number_parse_exponent_zero_test:N #1
296 {
297     \quark_if_recursion_tail_stop_do:Nn #1
298     { \tl_set:Nn \l__siunitx_number_exponent_tl { { } 0 } }
299     \str_if_eq:nnTF {#1} { 0 }
300     { \__siunitx_number_parse_exponent_zero_test:N }
301     { \__siunitx_number_parse_exponent_check:N #1 }
302 }
303 \cs_new_protected:Npn \__siunitx_number_parse_exponent_check:N #1
304 {
305     \quark_if_recursion_tail_stop:N #1
306     \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
307     {
308         \tl_put_right:Nn \l__siunitx_number_exponent_tl {#1}
309         \__siunitx_number_parse_exponent_check:N
310     }
311     { \__siunitx_number_parse_exponent_cleanup:wN }
312 }
313 \cs_new_protected:Npn \__siunitx_number_parse_exponent_cleanup:wN
314 #1 \q_recursion_stop
315 { \tl_clear:N \l__siunitx_number_parsed_tl }

```

There are two parts to the replacement code. First, any active hyphens signs are normalised: these can come up with some packages and cause issues. Multi-token signs then are converted to the single token equivalents so that everything else can work on a one token basis.

```

\__siunitx_number_parse_replace:
\__siunitx_number_parse_replace_aux:nN
\__siunitx_number_parse_replace_sign:
\__siunitx_number_parse_sign_replacement_tl

```

```

316 \cs_new_protected:Npn \__siunitx_number_parse_replace:
317 {
318   \__siunitx_number_parse_replace_minus:
319   \exp_last_unbraced:NV \__siunitx_number_parse_replace_aux:nN
320   \c__siunitx_number_parse_sign_replacement_tl
321   { ? } \q_recursion_tail
322   \q_recursion_stop
323 }
324 \cs_set_protected:Npn \__siunitx_number_parse_replace_aux:nN #1#2
325 {
326   \quark_if_recursion_tail_stop:N #2
327   \tl_replace_all:Nnn \l__siunitx_number_arg_tl {#1} {#2}
328   \__siunitx_number_parse_replace_aux:nN
329 }
330 \tl_const:Nn \c__siunitx_number_parse_sign_replacement_tl
331 {
332   { -- } \mp
333   { +- } \pm
334   { << } \ll
335   { <= } \le
336   { >> } \gg
337   { >= } \ge
338 }
339 \group_begin:
340   \char_set_catcode_active:N \-
341   \cs_new_protected:Npx \__siunitx_number_parse_replace_minus:
342   {
343     \tl_replace_all:Nnn \exp_not:N \l__siunitx_number_arg_tl
344     { \exp_not:N - } { \token_to_str:N - }
345   }
346 \group_end:

```

(End definition for __siunitx_number_parse_exponent: and others.)

_siunitx_number_parse_finalise: Combine all of the bits of a number together: both the real and imaginary parts contain
_siunitx_number_parse_finalise:nw all of the data.

```

347 \cs_new_protected:Npn \__siunitx_number_parse_finalise:
348 {
349   \tl_if_empty:NF \l__siunitx_number_parsed_tl
350   {
351     \tl_set:Nx \l__siunitx_number_parsed_tl
352     {
353       { \exp_not:N \l__siunitx_number_comparator_tl }
354       \exp_not:N \l__siunitx_number_parsed_tl
355       \exp_after:wN \__siunitx_number_parse_finalise:nw
356       \l__siunitx_number_exponent_tl \q_stop
357     }
358   }
359 }
360 \cs_new:Npn \__siunitx_number_parse_finalise:nw #1#2 \q_stop
361 {
362   { \exp_not:n {#1} }
363   { \exp_not:n {#2} }
364 }

```

(End definition for `__siunitx_number_parse_finalise:` and `__siunitx_number_parse_finalise:nw`.)

`__siunitx_number_parse_loop:` At this stage, the partial input `\l__siunitx_number_arg_tl` will contain any mantissa, which may contain an uncertainty or complex part. Parsing this and allowing for all of the different formats possible is best done using a token-by-token approach. However, as at each stage only a subset of tokens are valid, the approach take is to use a set of semi-dedicated functions to parse different components along with switches to allow a sensible amount of code sharing.

```

365 \cs_new_protected:Npn \__siunitx_number_parse_loop:
366 {
367   \tl_clear:N \l__siunitx_number_partial_tl
368   \exp_after:wN \__siunitx_number_parse_loop_first:NNN
369   \exp_after:wN \l__siunitx_number_parsed_tl \exp_after:wN \c_true_bool
370   \l__siunitx_number_arg_tl
371   \q_recursion_tail \q_recursion_stop
372 }

```

The very first token of the input is handled with a dedicated function. Valid cases here are

- Entirely blank if the original input was for example `+e10`: simply clean up if in the integer part of issue an error if in a second part (complex number, *etc.*).
- An integer part digit: pass through to the main collection routine.
- A decimal marker: store an empty integer part and move to the main collection routine for a decimal part.

Anything else is invalid and sends the code to the abort function.

```

373 \cs_new_protected:Npn \__siunitx_number_parse_loop_first:NNN #1#2#3
374 {
375   \quark_if_recursion_tail_stop_do:Nn #3
376   {
377     \bool_if:NTF #2
378     { \tl_put_right:Nn #1 { { 1 } { } { } } }
379     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
380   }
381   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#3}
382   {
383     \__siunitx_number_parse_loop_main:NNNNN
384     #1 \c_true_bool \c_false_bool #2 #3
385   }
386   {
387     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#3}
388     {
389       \tl_put_right:Nn #1 { { 0 } }
390       \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
391     }
392     { \__siunitx_number_parse_loop_break:wN }
393   }
394 }

```

A single function is used to cover the “main” part of numbers: finding real, complex or separated uncertainty parts and covering both the integer and decimal components. This works because these elements share a lot of concepts: a small number of switches can

be used to differentiate between them. To keep the code at least somewhat readable, this main function deals with the validity testing but hands off other tasks to dedicated auxiliaries for each case.

The possibilities are

- The number terminates, meaning that some digits were collected and everything is simply tidied up (as far as the loop is concerned).
- A digit is found: this is the common case and leads to a storage auxiliary (which handles non-significant zeros).
- A decimal marker is found: only valid in the integer part and there leading to a store-and-switch situation.
- An open-uncertainty token: switch to the dedicated collector for uncertainties.
- A sign token (if allowed): stop collecting this number and restart collection for the second part.

```

395 \cs_new_protected:Npn \__siunitx_number_parse_loop_main:NNNN #1#2#3#4#5
396 {
397   \quark_if_recursion_tail_stop_do:Nn #5
398   { \__siunitx_number_parse_loop_main_end:NN #1#2 }
399   \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
400   { \__siunitx_number_parse_loop_main_digit:NNNN #1#2#3#4#5 }
401   {
402     \tl_if_in:NnTF \l__siunitx_number_input_decimal_tl {#5}
403     {
404       \bool_if:NTF #2
405       { \__siunitx_number_parse_loop_main_decimal:NN #1 #4 }
406       { \__siunitx_number_parse_loop_break:wN }
407     }
408     {
409       \tl_if_in:NnTF \l__siunitx_number_input_uncert_open_tl {#5}
410       { \__siunitx_number_parse_loop_main_uncert:NNN #1#2 #4 }
411       {
412         \bool_if:NTF #4
413         {
414           \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#5}
415           {
416             \__siunitx_number_parse_loop_main_sign:NNN
417             #1#2 #5
418           }
419           { \__siunitx_number_parse_loop_break:wN }
420         }
421         { \__siunitx_number_parse_loop_break:wN }
422       }
423     }
424   }
425 }
```

If the main loop finds the end marker then there is a tidy up phase. The current partial number is stored either as the integer or decimal, depending on the setting for the indicator switch. For the integer part, if no number has been collected then one or more

non-significant zeros have been dropped. Exactly one zero is therefore needed to make sure the parsed result is correct.

```

426 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_end:NN #1#2
427 {
428   \bool_lazy_and:nnT
429     {#2} { \tl_if_empty_p:N \l__siunitx_number_partial_tl }
430     { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
431   \tl_put_right:Nx #1
432     {
433       { \exp_not:V \l__siunitx_number_partial_tl }
434       \bool_if:NT #2 { { } }
435       { }
436     }
437 }

```

The most common case for the main loop collector is to find a digit. Here, in the integer part it is possible that zeros are non-significant: that is handled using a combination of a switch and a string test. Other than that, the situation here is simple: store the input and loop.

```

438 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_digit:NNNN #1#2#3#4#5
439 {
440   \bool_lazy_or:nnTF
441     {#3} { ! \str_if_eq_p:nn {#5} { 0 } }
442     {
443       \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
444       \__siunitx_number_parse_loop_main:NNNN #1 #2 \c_true_bool #4
445     }
446     { \__siunitx_number_parse_loop_main:NNNN #1 #2 \c_false_bool #4 }
447 }

```

When a decimal marker was found, move the integer part to the store and then go back to the loop with the flags set correctly. There is the case of non-significant zeros to cover before that, of course.

```

448 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_decimal:NN #1#2
449 {
450   \__siunitx_number_parse_loop_main_store:NNN #1 \c_false_bool \c_false_bool
451   \__siunitx_number_parse_loop_after_decimal:NNN #1 #2
452 }

```

Starting an uncertainty part means storing the number to date as in other cases, with the possibility of a blank decimal part allowed for. The uncertainty itself is collected by a dedicated function as it is extremely restricted.

```

453 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_uncert:NNN #1#2#3
454 {
455   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_false_bool
456   \__siunitx_number_parse_loop_uncert:NNNN
457     #1 \c_true_bool \c_false_bool #3
458 }

```

If a sign is found, terminate the current number, store the sign as the first token of the second part and go back to do the dedicated first-token function.

```

459 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_sign:NNN #1#2#3
460 {
461   \__siunitx_number_parse_loop_main_store:NNN #1 #2 \c_true_bool
462   \tl_set:Nn \l__siunitx_number_flex_tl { {#3} }

```

```

463     \__siunitx_number_parse_loop_first:NNN
464     \l__siunitx_number_flex_tl \c_false_bool
465 }

```

A common auxiliary for the various non-digit token functions: tidy up the integer and decimal parts of a number. Here, the two flags are used to indicate if empty decimal and uncertainty parts should be included in the storage cycle.

```

466 \cs_new_protected:Npn \__siunitx_number_parse_loop_main_store:NNN #1#2#3
467 {
468     \tl_if_empty:NT \l__siunitx_number_partial_tl
469     { \tl_set:Nn \l__siunitx_number_partial_tl { 0 } }
470     \tl_put_right:Nx #1
471     {
472         { \exp_not:V \l__siunitx_number_partial_tl }
473         \bool_if:NT #2 { { } }
474         \bool_if:NT #3 { { } }
475     }
476     \tl_clear:N \l__siunitx_number_partial_tl
477 }

```

After a decimal marker there has to be a digit if there wasn't one before it. That is handled by using a dedicated function, which checks for an empty integer part first then either simply hands off or looks for a digit.

```

478 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_decimal:NNN #1#2#3
479 {
480     \tl_if_blank:fTF { \exp_after:wN \use_none:n #1 }
481     {
482         \quark_if_recursion_tail_stop_do:Nn #3
483         { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
484         \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#1}
485         {
486             \tl_put_right:Nn \l__siunitx_number_partial_tl {#3}
487             \__siunitx_number_parse_loop_main:NNNNN
488             #1 \c_false_bool \c_true_bool #2
489         }
490         { \__siunitx_number_parse_loop_break:wN }
491     }
492     {
493         \__siunitx_number_parse_loop_main:NNNNN
494         #1 \c_false_bool \c_true_bool #2 #3
495     }
496 }

```

Inside the brackets for an uncertainty the range of valid choices is very limited. Either the token is a digit, in which case there is a test to look for non-significant zeros, or it is a closing bracket. The latter is not valid for the very first token, which is handled using a switch (it's a simple enough difference).

```

497 \cs_new_protected:Npn \__siunitx_number_parse_loop_uncert:NNNNN #1#2#3#4#5
498 {
499     \quark_if_recursion_tail_stop_do:Nn #5
500     { \__siunitx_number_parse_loop_break:wN \q_recursion_stop }
501     \tl_if_in:NnTF \l__siunitx_number_input_digit_tl {#5}
502     {
503         \bool_lazy_or:nnTF
504         {#3} { ! \str_if_eq_p:nn {#5} { 0 } }

```

```

505     {
506       \tl_put_right:Nn \l__siunitx_number_partial_tl {#5}
507       \__siunitx_number_parse_loop_uncert:NNNNN
508       #1 \c_false_bool \c_true_bool #4
509     }
510     {
511       \__siunitx_number_parse_loop_uncert:NNNNN
512       #1 \c_false_bool \c_false_bool #4
513     }
514   }
515   {
516     \tl_if_in:NnTF \l__siunitx_number_input_uncert_close_tl {#5}
517     {
518       \bool_if:NTF #2
519       { \__siunitx_number_parse_loop_break:wN }
520       {
521         \tl_if_empty:NTF \l__siunitx_number_partial_tl
522         { \tl_put_right:Nx #1 { { } } }
523         {
524           \tl_set:Nx \l__siunitx_number_partial_tl
525           { { S } { \exp_not:V \l__siunitx_number_partial_tl } }
526           \__siunitx_number_parse_loop_main_store:NNN #1
527           \c_false_bool \c_false_bool
528         }
529         \__siunitx_number_parse_loop_after_uncert:N
530       }
531     }
532     { \__siunitx_number_parse_loop_break:wN }
533   }
534 }

```

No further tokens are allowed after an uncertainty in parenthesis.

```

535 \cs_new_protected:Npn \__siunitx_number_parse_loop_after_uncert:N #1
536 {
537   \quark_if_recursion_tail_stop:N #1
538   \__siunitx_number_parse_loop_break:wN
539 }

```

Something is not right: remove all of the remaining tokens from the number and clear the storage areas as a signal for the next part of the code.

```

540 \cs_new_protected:Npn \__siunitx_number_parse_loop_break:wN
541 #1 \q_recursion_stop
542 {
543   \tl_clear:N \l__siunitx_number_flex_tl
544   \tl_clear:N \l__siunitx_number_parsed_tl
545 }

```

(End definition for `__siunitx_number_parse_loop:` and others.)

`__siunitx_number_parse_sign:`
`__siunitx_number_parse_sign_aux:Nw`

The first token of a number after a comparator could be a sign. A quick check is made and if found stored. For the number to be valid it has to be more than just a sign, so the next part of the chain is only called if that is the case.

```

546 \cs_new_protected:Npn \__siunitx_number_parse_sign:
547 {
548   \exp_after:wN \__siunitx_number_parse_sign_aux:Nw

```

```

549     \l__siunitx_number_arg_tl \q_stop
550   }
551 \cs_new_protected:Npn \__siunitx_number_parse_sign_aux:Nw #1#2 \q_stop
552 {
553   \tl_if_in:NnTF \l__siunitx_number_input_sign_tl {#1}
554   {
555     \tl_set:Nn \l__siunitx_number_arg_tl {#2}
556     \bool_lazy_and:nnTF
557     { \token_if_eq_charcode_p:NN #1 + }
558     { ! \l__siunitx_number_explicit_plus_bool }
559     { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
560     { \tl_set:Nn \l__siunitx_number_parsed_tl { {#1} } }
561   }
562   { \tl_set:Nn \l__siunitx_number_parsed_tl { { } } }
563   \tl_if_empty:NNTF \l__siunitx_number_arg_tl
564   { \tl_clear:N \l__siunitx_number_parsed_tl }
565   { \__siunitx_number_parse_exponent: }
566 }

```

(End definition for __siunitx_number_parse_sign: and __siunitx_number_parse_sign_aux:Nw.)

2.4 Processing numbers

```

\l__siunitx_number_drop_exponent_bool
\l__siunitx_number_drop_uncertainty_bool
\l__siunitx_number_drop_zero_decimal_bool
\l__siunitx_number_exponent_mode_tl
\l__siunitx_number_exponent_fixed_int
\l__siunitx_number_min_decimal_int
\l__siunitx_number_min_integer_int
\l__siunitx_number_round_half_even_bool
\l__siunitx_number_round_min_tl
\l__siunitx_number_round_mode_tl
\l__siunitx_number_round_pad_bool
\l__siunitx_number_round_precision_int

567 \keys_define:nn { siunitx }
568 {
569   drop-exponent .bool_set:N =
570     \l__siunitx_number_drop_exponent_bool ,
571   drop-uncertainty .bool_set:N =
572     \l__siunitx_number_drop_uncertainty_bool ,
573   drop-zero-decimal .bool_set:N =
574     \l__siunitx_number_drop_zero_decimal_bool ,
575   exponent-mode .choices:nn =
576     { engineering , fixed , input , scientific }
577     { \tl_set_eq:NN \l__siunitx_number_exponent_mode_tl \l_keys_choice_tl } ,
578   fixed-exponent .int_set:N =
579     \l__siunitx_number_exponent_fixed_int ,
580   minimum-decimal-digits .int_set:N =
581     \l__siunitx_number_min_decimal_int ,
582   minimum-integer-digits .int_set:N =
583     \l__siunitx_number_min_integer_int ,
584   round-half .choice: ,
585   round-half / even .code:n =
586     { \bool_set_true:N \l__siunitx_number_round_half_even_bool } ,
587   round-half / up .code:n =
588     { \bool_set_false:N \l__siunitx_number_round_half_even_bool } ,
589   round-minimum .tl_set:N =
590     \l__siunitx_number_round_min_tl ,
591   round-mode .choices:nn =
592     { figures , none , places , uncertainty }
593     { \tl_set_eq:NN \l__siunitx_number_round_mode_tl \l_keys_choice_tl } ,
594   round-pad .bool_set:N =
595     \l__siunitx_number_round_pad_bool ,
596   round-precision .int_set:N =

```

```

597     \l__siunitx_number_round_precision_int ,
598   }
599   \bool_new:N \l__siunitx_number_round_half_even_bool
600   \tl_new:N \l__siunitx_number_exponent_mode_tl
601   \tl_new:N \l__siunitx_number_round_mode_tl

```

(End definition for \l__siunitx_number_drop_exponent_bool and others.)

\siunitx_number_process:NN

A top-level interface for the processing tools.

```

\__siunitx_number_process:nnnnnnn
602 \cs_new_protected:Npn \siunitx_number_process:NN #1#2
603 {
604   \tl_if_empty:NF #1
605   {
606     \__siunitx_number_drop_uncertainty:NN #1 #2
607     \exp_after:wN \__siunitx_number_process:nnnnnnn #2 #2 #2
608     \__siunitx_number_drop_exponent:NN #2 #2
609     \__siunitx_number_zero_decimal:NN #2 #2
610     \__siunitx_number_digits:NN #2 #2
611   }
612 }
613 \cs_new_protected:Npn \__siunitx_number_process:nnnnnnn #1#2#3#4#5#6#7#8#9
614 {
615   \bool_lazy_and:nnF
616   { \str_if_eq_p:nn {#3} { 0 } }
617   {
618     \str_if_eq_p:ee
619     { \exp_not:n {#4} } { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
620   }
621   {
622     \__siunitx_number_exponent:NN #8 #9
623     \__siunitx_number_round:NN #9 #9
624   }
625 }

```

(End definition for \siunitx_number_process:NN and __siunitx_number_process:nnnnnnn. This function is documented on page 21.)

```

\__siunitx_number_exponent:NN
\siunitx_number_exponent_engineering:nnnnnnn
\__siunitx_number_exponent_fixed:nnnnnnn
\__siunitx_number_exponent_input:nnnnnnn
\__siunitx_number_exponent_scientific:nnnnnnn
\__siunitx_number_exponent_scientific:nnnw
\__siunitx_number_exponent_shift:nnn
\__siunitx_number_exponent_shift:nnf
\__siunitx_number_exponent_shift_down:nnnw
\__siunitx_number_exponent_shift_down:nnn
\__siunitx_number_exponent_shift_down:nw
\__siunitx_number_exponent_shift_up:nnn
\__siunitx_number_exponent_shift_up:nnw
\__siunitx_number_exponent_shift_up_aux:nnn
\__siunitx_number_exponent_finalise:n
\siunitx_number_exponent_engineering_aux:nnnnnnn
\__siunitx_number_exponent_engineering_0:nnnn
\__siunitx_number_exponent_engineering_1:nnnn
\__siunitx_number_exponent_engineering_2:nnnn
\__siunitx_number_exponent_engineering:nW

```

Manipulating an exponent is done using a single expansion function *unless* dealing with engineering-style output. The latter is easier to handle by first converting to scientific output, then post-processing. (Once e-type expansion is generally available, this will be handling using a single \tl_set:Nx.)

```

626 \cs_new_protected:Npn \__siunitx_number_exponent:NN #1#2
627 {
628   \tl_set:Nx #2
629   {
630     \cs:w
631     \__siunitx_number_exponent_ \l__siunitx_number_exponent_mode_tl :nnnnnnn
632     \exp_after:wN
633     \cs_end: #1
634   }
635   \str_if_eq:VnT \l__siunitx_number_exponent_mode_tl { engineering }
636   {
637     \tl_set:Nx #1
638     { \exp_after:wN \__siunitx_number_exponent_engineering_aux:nnnnnnn #1 }

```

```

639     }
640   }
641   \cs_new:Npn \__siunitx_number_exponent_fixed:nnnnnnn #1#2#3#4#5#6#7
642   {
643     \exp_not:n { {#1} {#2} }
644     \exp_args:Nf \__siunitx_number_exponent_shift:nnn
645       { \int_eval:n { \l__siunitx_number_exponent_fixed_int - (#6#7) } }
646       {#3} {#4}
647     \exp_not:n { {#5} }
648     \exp_not:n { {#6} } { \int_use:N \l__siunitx_number_exponent_fixed_int }
649   }
650   \cs_new:Npn \__siunitx_number_exponent_input:nnnnnnn #1#2#3#4#5#6#7
651   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

To convert to scientific notation, the key question is to find the number of significant places. That is easy enough if the number has a non-zero integer component. For a pure decimal, we have to trim off leading zeros in a loop.

```

652   \cs_new:Npn \__siunitx_number_exponent_scientific:nnnnnnn #1#2#3#4#5#6#7
653   {
654     \exp_not:n { {#1} {#2} }
655     \int_compare:nNnTF { \tl_count:n {#3} } = 1
656     {
657       \str_if_eq:nnTF {#3} { 0 }
658       {
659         \__siunitx_number_exponent_scientific:nnnw
660           { 0 } {#5} { #6#7 } #4 \q_stop
661       }
662       { \exp_not:n { {#3} {#4} {#5} {#6} {#7} } }
663     }
664     {
665       \__siunitx_number_exponent_shift:nnn { \tl_count:n {#3} - 1 } {#3} {#4}
666       \exp_not:n { {#5} }
667       \__siunitx_number_exponent_finalise:n { \tl_count:n {#3} + #6#7 - 1 }
668     }
669   }
670   \cs_new_eq:NN \__siunitx_number_exponent_engineering:nnnnnnn
671   \__siunitx_number_exponent_scientific:nnnnnnn
672   \cs_new:Npn \__siunitx_number_exponent_scientific:nnnw #1#2#3#4#5 \q_stop
673   {
674     \str_if_eq:nnTF {#4} { 0 }
675     {
676       \__siunitx_number_exponent_scientific:nnnw
677         { #1 - 1 } {#2} {#3} #5 \q_stop
678     }
679     {
680       \exp_not:n { {#4} {#5} {#2} }
681       \__siunitx_number_exponent_finalise:n { #1 + #3 - 1 }
682     }
683   }

```

When adjusting the exponent position, there are two paths depending on which way the shift takes place.

```

684   \cs_new:Npn \__siunitx_number_exponent_shift:nnn #1#2#3
685   {
686     \int_compare:nNnTF {#1} > 0

```

```

687 { \_siunitx\_number\_exponent\_shift\_down:nnnw {#1} {#3} { } #2 \q\_stop }
688 {
689   \int\_compare:nNnTF {#1} < 0
690   { \_siunitx\_number\_exponent\_shift\_up:nnn {#1} {#2} {#3} }
691   { {#2} {#3} }
692 }
693 }
694 \cs\_generate\_variant:Nn \_siunitx\_number\_exponent\_shift:nnn { nnf }

```

For shifting the exponent down, there is first a loop to reserve the integer part before doing the work: that of course has to be undone for any remainder at the end of the process.

```

695 \cs\_new:Npn \_siunitx\_number\_exponent\_shift\_down:nnnw #1#2#3#4#5 \q\_stop
696 {
697   \tl\_if\_blank:nTF {#5}
698   { \_siunitx\_number\_exponent\_shift\_down:nnn {#1} { #4 #3 } {#2} }
699   { \_siunitx\_number\_exponent\_shift\_down:nnnw {#1} {#2} { #4 #3 } #5 \q\_stop }
700 }
701 \cs\_new:Npn \_siunitx\_number\_exponent\_shift\_down:nnn #1#2#3
702 {
703   \int\_compare:nNnTF {#1} = 0
704   { { \tl\_reverse:n {#2} } \exp\_not:n { {#3} } }
705   { \_siunitx\_number\_exponent\_shift\_down:nw {#1} #2 \q\_stop {#3} }
706 }
707 \cs\_new:Npn \_siunitx\_number\_exponent\_shift\_down:nw #1#2#3 \q\_stop #4
708 {
709   \tl\_if\_blank:nTF {#3}
710   { \_siunitx\_number\_exponent\_shift\_down:nnn { #1 - 1 } { 0 } { #2#4 } }
711   { \_siunitx\_number\_exponent\_shift\_down:nnn { #1 - 1 } {#3} { #2#4 } }
712 }

```

For shifting the exponent up, we can run out of decimal digits, at which point filling is easy. Other than that a simple loop as we are picking input off the front of the decimal part.

```

713 \cs\_new:Npn \_siunitx\_number\_exponent\_shift\_up:nnn #1#2#3
714 {
715   \tl\_if\_blank:nTF {#3}
716   { \_siunitx\_number\_exponent\_shift\_up\_aux:nnn { #1 + 1 } { #2 0 } { } }
717   { \_siunitx\_number\_exponent\_shift\_up:nnw {#1} {#2} #3 \q\_stop }
718 }
719 \cs\_new:Npn \_siunitx\_number\_exponent\_shift\_up:nnw #1#2#3#4 \q\_stop
720 { \_siunitx\_number\_exponent\_shift\_up\_aux:nnn { #1 + 1 } { #2 #3 } {#4} }
721 \cs\_new:Npn \_siunitx\_number\_exponent\_shift\_up\_aux:nnn #1#2#3
722 {
723   \int\_compare:nNnTF {#1} = 0
724   { \exp\_not:n { {#2} {#3} } }
725   {
726     \tl\_if\_blank:nTF {#3}
727     {
728       {
729         \exp\_not:n {#2}
730         \prg\_replicate:nn { \int\_abs:n {#1} } { 0 }
731       }
732     { }
733   }

```

```

734         { \_siunitx_number_exponent_shift_up:nnn {#1} {#2} {#3} }
735     }
736 }

```

Tidy up the exponent to put the sign in the right place.

```

737 \cs_new:Npn \_siunitx_number_exponent_finalise:n #1
738 {
739     \int_compare:nNnTF {#1} > 0
740     { { } }
741     { { - } }
742     { \int_abs:n {#1} }
743 }

```

This could (and eventually will) be combined with the main function above: that will need **e**-type expansion. The input has already been normalised such that the integer part is in the range $1 \leq n < 10$. Thus there are only three cases to deal with, depending on the required adjustment to the exponent.

```

744 \cs_new:Npn \_siunitx_number_exponent_engineering_aux:nnnnnnn #1#2#3#4#5#6#7
745 {
746     \exp_not:n { {#1} {#2} }
747     \use:c
748     {
749         \_siunitx_number_exponent_engineering_
750         \int_compare:nNnTF {#6#7} < 0
751         {
752             \int_case:nnF { \int_mod:nn { #7 } { 3 } }
753             {
754                 { 1 } { 2 }
755                 { 2 } { 1 }
756             }
757             { 0 }
758         }
759         { \int_mod:nn {#7} { 3 } }
760         :nnnn
761     }
762     {#3} {#4} {#5} {#6#7}
763 }
764 \cs_new:cpn { \_siunitx_number_exponent_engineering_0:nnnn } #1#2#3#4
765 {
766     \exp_not:n { {#1} {#2} {#3} }
767     \_siunitx_number_exponent_finalise:n {#4}
768 }
769 \cs_new:cpn { \_siunitx_number_exponent_engineering_1:nnnn } #1#2#3#4
770 {
771     \tl_if_blank:nTF {#2}
772     { { \exp_not:n { #1 0 } } { } }
773     {
774         { \exp_not:n {#1} \exp_not:o { \tl_head:w #2 \q_stop } }
775         { \exp_not:f { \tl_tail:n {#2} } }
776     }
777     \exp_not:n { {#3} }
778     \_siunitx_number_exponent_finalise:n { #4 - 1 }
779 }
780 \cs_new:cpn { \_siunitx_number_exponent_engineering_2:nnnn } #1#2#3#4
781 {

```



```

782 \tl_if_blank:nTF {#2}
783 { { \exp_not:n { #1 00 } } { } }
784 { \__siunitx_number_exponent_engineering:nNw {#1} #2 \q_stop }
785 \exp_not:n { {#3} }
786 \__siunitx_number_exponent_finalise:n { #4 - 2 }
787 }
788 \cs_new:Npn \__siunitx_number_exponent_engineering:nNw #1#2#3 \q_stop
789 {
790 \tl_if_blank:nTF {#3}
791 { { \exp_not:n { #1#2 0 } } { } }
792 {
793 { \exp_not:n {#1#2} \exp_not:o { \tl_head:w #3 \q_stop } }
794 { \exp_not:f { \tl_tail:n {#3} } }
795 }
796 }

```

(End definition for `__siunitx_number_exponent:NN` and others.)

```

\__siunitx_number_digits:NN
\__siunitx_number_digits:nnnnnnn
\__siunitx_number_digits:Nn

```

Forcing a minimum number of digits in each part is quite easy. As the common case is that we don't do anything here, there is no real need to optimise the calculation (normally also numbers have only a few digits).

```

797 \cs_new_protected:Npn \__siunitx_number_digits:NN #1#2
798 {
799 \tl_set:Nx #2
800 { \exp_after:wN \__siunitx_number_digits:nnnnnnn #1 }
801 }
802 \cs_new:Npn \__siunitx_number_digits:nnnnnnn #1#2#3#4#5#6#7
803 {
804 \exp_not:n { {#1} {#2} }
805 {
806 \__siunitx_number_digits:Nn \l__siunitx_number_min_integer_int {#3}
807 \exp_not:n {#3}
808 }
809 {
810 \exp_not:n {#4}
811 \__siunitx_number_digits:Nn \l__siunitx_number_min_decimal_int {#4}
812 }
813 \exp_not:n { {#5} {#6} {#7} }
814 }
815 \cs_new:Npn \__siunitx_number_digits:Nn #1#2
816 {
817 \int_compare:nNnT
818 { #1 - \tl_count:n {#2} } > 0
819 { \prg_replicate:nn { #1 - \tl_count:n {#2} } { 0 } }
820 }

```

(End definition for `__siunitx_number_digits:NN`, `__siunitx_number_digits:nnnnnnn`, and `__siunitx_number_digits:Nn`.)

```

\__siunitx_number_drop_exponent:NN
\__siunitx_number_drop_exponent:nnnnnnn

```

Simple stripping of the exponent.

```

821 \cs_new_protected:Npn \__siunitx_number_drop_exponent:NN #1#2
822 {
823 \bool_if:NT \l__siunitx_number_drop_exponent_bool
824 {

```

```

825         \tl_set:Nx #2
826         { \exp_after:wN \__siunitx_number_drop_exponent:nnnnnnn #1 }
827     }
828 }
829 \cs_new:Npn \__siunitx_number_drop_exponent:nnnnnnn #1#2#3#4#5#6#7
830 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} { } { } { 0 } } }

```

(End definition for __siunitx_number_drop_exponent:NN and __siunitx_number_drop_exponent:nnnnnnn.)

__siunitx_number_drop_uncertainty:NN
__siunitx_number_drop_uncertainty:nnnnnnn

Simple stripping of the uncertainty.

```

831 \cs_new_protected:Npn \__siunitx_number_drop_uncertainty:NN #1#2
832 {
833     \bool_if:NT \l__siunitx_number_drop_uncertainty_bool
834     {
835         \tl_set:Nx #2
836         { \exp_after:wN \__siunitx_number_drop_uncertainty:nnnnnnn #1 }
837     }
838 }
839 \cs_new:Npn \__siunitx_number_drop_uncertainty:nnnnnnn #1#2#3#4#5#6#7
840 { \exp_not:n { {#1} {#2} {#3} {#4} { } {#6} {#7} } }

```

(End definition for __siunitx_number_drop_uncertainty:NN and __siunitx_number_drop_uncertainty:nnnnnnn.)

__siunitx_number_round:NN
__siunitx_number_round_none:nnnnnnn

Rounding is at the top level simple enough: fire off the expandable set up which does the work.

```

841 \cs_new_protected:Npn \__siunitx_number_round:NN #1#2
842 {
843     \tl_set:Nx #2
844     {
845         \cs:w
846         __siunitx_number_round_ \l__siunitx_number_round_mode_tl :nnnnnnn
847         \exp_after:wN
848         \cs_end: #1
849     }
850 }
851 \cs_new:Npn \__siunitx_number_round_none:nnnnnnn #1#2#3#4#5#6#7
852 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }

```

(End definition for __siunitx_number_round:NN and __siunitx_number_round_none:nnnnnnn.)

__siunitx_number_round:nnn
__siunitx_number_round:fnn

Actually doing the rounding needs us to work from the least significant digit, so we start by reversing the input. We *could* also drop digits in this phase, but tracking everything would be horrible, so we go slightly slower but clearer and split the steps. First we reverse the decimal part, then the integer.

__siunitx_number_round_auxi:nnnN
__siunitx_number_round_auxii:nnnN
__siunitx_number_round_auxiii:nnnN
__siunitx_number_round_auxiv:nnN
__siunitx_number_round_auxv:nnN
__siunitx_number_round_auxvi:nnN
__siunitx_number_round_auxvii:nnN
__siunitx_number_round_auxviii:nnN
__siunitx_number_round_final_integer:nnw
__siunitx_number_round_final_decimal:nnw
__siunitx_number_round_final_output:nn
__siunitx_number_round_final_output:ff
__siunitx_number_round_final:nn
__siunitx_number_round_final:fn
__siunitx_number_round_final_shift:nn
__siunitx_number_round_final_shift:ff
__siunitx_number_round_final_shift:Nw
__siunitx_number_round_engineering:nn
__siunitx_number_round_fixed:nn
__siunitx_number_round_input:nn
__siunitx_number_round_scientific:nn
__siunitx_number_round_engineering:NNNNn

```

853 \cs_new:Npn \__siunitx_number_round:nnn #1#2#3
854 {
855     \__siunitx_number_round_auxi:nnnN {#1} {#2} { }
856     #3 \q_recursion_tail \q_recursion_stop
857 }
858 \cs_generate_variant:Nn \__siunitx_number_round:nnn { f }
859 \cs_new:Npn \__siunitx_number_round_auxi:nnnN #1#2#3#4
860 {
861     \quark_if_recursion_tail_stop_do:Nn #4
862     {

```

```

863         \__siunitx_number_round_auxii:nnnN {#1} {#3} { } #2
864         \q_recursion_tail \q_recursion_stop
865     }
866     \__siunitx_number_round_auxi:nnnN {#1} {#2} {#4#3}
867 }
868 \cs_new:Npn \__siunitx_number_round_auxii:nnnN #1#2#3#4
869 {
870     \quark_if_recursion_tail_stop_do:Nn #4
871     {
872         \tl_if_blank:nTF {#2}
873         {
874             \__siunitx_number_round_auxiv:nnnN {#1} { } { } #3
875             \q_recursion_tail \q_recursion_stop
876         }
877         {
878             \__siunitx_number_round_auxiii:nnnN {#1} {#3} { } #2
879             \q_recursion_tail \q_recursion_stop
880         }
881     }
882     \__siunitx_number_round_auxii:nnnN {#1} {#2} {#4#3}
883 }

```

We now have the input reversed plus how many digits we need to discard (#1). We have two functions, one which deals with the decimal part, one of which deals with the integer. In the latter, we should never hit the end before we've dropped all the digits: the fixed-zero is a fall-back in case something weird happens. For the integer case, we need to collect up zeros to pad the length back out correctly later.

```

884 \cs_new:Npn \__siunitx_number_round_auxiii:nnnN #1#2#3#4
885 {
886     \quark_if_recursion_tail_stop_do:Nn #4
887     {
888         \__siunitx_number_round_auxiv:nnnN {#1} { } {#3} #2
889         \q_recursion_tail \q_recursion_stop
890     }
891     \int_compare:nNnTF {#1} > 0
892     {
893         \exp_args:Nf \__siunitx_number_round_auxiii:nnnN
894         { \int_eval:n { #1 - 1 } } {#2} { #4#3 }
895     }
896     { \__siunitx_number_round_auxv:nnN {#3} {#2} #4 }
897 }
898 \cs_new:Npn \__siunitx_number_round_auxiv:nnnN #1#2#3#4
899 {
900     \quark_if_recursion_tail_stop_do:Nn #4
901     { { 0 } { } }
902     \int_compare:nNnTF {#1} > 0
903     {
904         \exp_args:Nf \__siunitx_number_round_auxiv:nnnN
905         { \int_eval:n { #1 - 1 } } { #2 0 } { #4#3 }
906     }
907     { \__siunitx_number_round_auxvi:nnnN {#3} {#2} #4 }
908 }

```

The lead off to rounding proper needs to deal with the half-even rule: it can only apply

at this stage, when the *discarded* value can be exactly half.

```

909 \cs_new:Npn \__siunitx_number_round_auxv:nnN #1#2#3
910 {
911   \quark_if_recursion_tail_stop_do:Nn #3
912   {
913     \__siunitx_number_round_auxvi:nnN
914     {#1} { } #2 \q_recursion_tail \q_recursion_stop
915   }
916   \bool_lazy_or:nnTF
917   { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
918   {
919     \bool_lazy_all_p:n
920     {
921       { \l__siunitx_number_round_half_even_bool }
922       { \int_if_odd_p:n {#3} }
923       { \__siunitx_number_round_if_half_p:n {#1} }
924     }
925   }
926   { \__siunitx_number_round_final_decimal:nnw }
927   { \__siunitx_number_round_auxvii:nnN }
928   {#2} { } #3
929 }
930 \cs_new:Npn \__siunitx_number_round_auxvi:nnnN #1#2#3
931 {
932   \quark_if_recursion_tail_stop_do:Nn #3
933   { { 0 } { } }
934   \bool_lazy_or:nnTF
935   { \int_compare_p:nNn { 0 \tl_head:n {#1} } < 5 }
936   {
937     \bool_lazy_all_p:n
938     {
939       { \l__siunitx_number_round_half_even_bool }
940       { \int_if_odd_p:n {#3} }
941       { \__siunitx_number_round_if_half_p:n {#1} }
942     }
943   }
944   { \__siunitx_number_round_final_integer:nnw }
945   { \__siunitx_number_round_auxviii:nnN }
946   { } {#2} #3
947 }

```

The main rounding routines. These are only ever called when there is rounding to do, so there is no need to carry a flag forward. Thus the question to ask is simple: is the next value a 9 or not (as that continues the sequence). There is a general need to handle the case where a zero is rounded up: that automatically means a need to trim the other end.

```

948 \cs_new:Npn \__siunitx_number_round_auxvii:nnN #1#2#3
949 {
950   \quark_if_recursion_tail_stop_do:Nn #3
951   {
952     \str_if_eq:nnTF {#1} { 0 }
953     {
954       \__siunitx_number_round_final_output:ff
955       { 1 }

```

```

956         { \_siunitx_number_round_truncate:n {#2} }
957     }
958     {
959         \_siunitx_number_round_auxviii:nnN {#2} { } #1
960         \q_recursion_tail \q_recursion_stop
961     }
962 }
963 \int_compare:nNnTF {#3} = 9
964 { \_siunitx_number_round_auxvii:nnN {#1} { 0 #2 } }
965 {
966     \int_compare:nNnTF {#3} = 0
967     {
968         \_siunitx_number_round_final_decimal:nnw
969         {#1} { 1 \_siunitx_number_round_truncate:n {#2} }
970     }
971     {
972         \_siunitx_number_round_final:fn
973         { \int_eval:n { #3 + 1 } }
974         { \_siunitx_number_round_final_decimal:nnw {#1} {#2} }
975     }
976 }
977 }
978 \cs_new:Npn \_siunitx_number_round_auxviii:nnN #1#2#3
979 {
980     \quark_if_recursion_tail_stop_do:Nn #3
981     {
982         \tl_if_blank:nTF {#1}
983         {
984             \_siunitx_number_round_final_shift:ff
985             {
986                 \exp_last_unbraced:Nf 1
987                 { \_siunitx_number_round_truncate_direct:n {#2} } 0
988             }
989             { }
990         }
991         {
992             \_siunitx_number_round_final_shift:ff
993             { 1 #2 }
994             { \_siunitx_number_round_truncate:n {#1} }
995         }
996     }
997     \int_compare:nNnTF {#3} = 9
998     { \_siunitx_number_round_auxviii:nnN {#1} { 0 #2 } }
999     {
1000         \_siunitx_number_round_final:fn
1001         { \int_eval:n { #3 + 1 } }
1002         { \_siunitx_number_round_final_integer:nnw {#1} {#2} }
1003     }
1004 }

```

Tidying up means grabbing the remaining digits and undoing the reversal.

```

1005 \cs_new:Npn \_siunitx_number_round_final_decimal:nnw
1006 #1#2#3 \q_recursion_tail \q_recursion_stop
1007 {
1008     \_siunitx_number_round_final_output:ff

```

```

1009     { \tl_reverse:n {#1} }
1010     { \tl_reverse:n {#3} #2 }
1011   }
1012 \cs_new:Npn \__siunitx_number_round_final_integer:nnw
1013   #1#2#3 \q_recursion_tail \q_recursion_stop
1014   {
1015     \__siunitx_number_round_final_output:ff
1016     { \tl_reverse:n {#3} #2 }
1017     {#1}
1018   }
1019 \cs_new:Npn \__siunitx_number_round_final_output:nn #1#2 { {#1} {#2} }
1020 \cs_generate_variant:Nn \__siunitx_number_round_final_output:nn { ff }
1021 \cs_new:Npn \__siunitx_number_round_final:nn #1#2
1022   { #2 #1 }
1023 \cs_generate_variant:Nn \__siunitx_number_round_final:nn { f }

```

Here we deal with the case where rounding applies along with an exponent set based on number of places. We can only get here if an additional integer digit has been added, so there is no need to test for that. There are two cases for action: when using `scientific` mode, where we always need to shift by one, and when using `engineering` mode if we now have four digits. The latter is a bit more work: we need to trim digits off as required.

```

1024 \cs_new:Npn \__siunitx_number_round_final_shift:nn #1#2
1025   {
1026     \str_if_eq:VnTF \l__siunitx_number_round_mode_tl { places }
1027     {
1028       \use:c
1029       { __siunitx_number_round_ \l__siunitx_number_exponent_mode_tl :nn }
1030       {#1} {#2}
1031     }
1032     { {#1} {#2} }
1033   }
1034 \cs_generate_variant:Nn \__siunitx_number_round_final_shift:nn { ff }
1035 \cs_new:Npn \__siunitx_number_round_engineering:nn #1#2
1036   {
1037     \int_compare:nNnTF { \tl_count:n {#1} } = 4
1038     {
1039       \__siunitx_number_round_engineering:NNNNn #1 {#2}
1040       { }
1041       \__siunitx_number_round_final_shift:Nw 3
1042     }
1043     { {#1} {#2} }
1044   }
1045 \cs_new:Npn \__siunitx_number_round_engineering:NNNNn #1#2#3#4#5
1046   {
1047     {#1}
1048     \exp_args:NV \__siunitx_number_round_engineering:nnN
1049     { \l__siunitx_number_round_precision_int } { }
1050     #2#3#4#5 \q_recursion_tail \q_recursion_stop
1051   }
1052 \cs_new:Npn \__siunitx_number_round_engineering:nnN #1#2#3
1053   {
1054     \quark_if_recursion_tail_stop_do:Nn #3 { {#2} }
1055     \int_compare:nNnTF {#1} = { 0 }
1056     { \use_i_delimit_by_q_recursion_stop:nw { {#2} } }

```

```

1057     { \_siunitx_number_round_engineering:nnN { #1 - 1 } { #2#3 } }
1058   }
1059 \cs_new:Npn \_siunitx_number_round_fixed:nn #1#2 { {#1} {#2} }
1060 \cs_new:Npn \_siunitx_number_round_input:nn #1#2 { {#1} {#2} }
1061 \cs_new:Npn \_siunitx_number_round_scientific:nn #1#2
1062   {
1063     \_siunitx_number_exponent_shift:nnf
1064     { 1 } {#1} { \_siunitx_number_round_truncate_direct:n {#2} }
1065   }
1066   \_siunitx_number_round_final_shift:Nw 1
1067 }
1068 \cs_new:Npn \_siunitx_number_round_final_shift:Nw #1#2 \_siunitx_number_round_places_end:n
1069   { \_siunitx_number_exponent_finalise:n { #3#4 + #1 } }

```

When we have rounded up to the next power of ten, we need to go back and remove one more digit. That only happens when rounding to a number of figures or when dealing with an integer part.

```

1070 \cs_new:Npn \_siunitx_number_round_truncate:n #1
1071   {
1072     \str_if_eq:VnTF \l_siunitx_number_round_mode_tl { figures }
1073     { \_siunitx_number_round_truncate_direct:n {#1} }
1074     {#1}
1075   }
1076 \cs_new:Npn \_siunitx_number_round_truncate_direct:n #1
1077   {
1078     \_siunitx_number_round_truncate:nnN { } { }
1079     #1 \q_recursion_tail \q_recursion_stop
1080   }
1081 \cs_new:Npn \_siunitx_number_round_truncate:nnN #1#2#3
1082   {
1083     \quark_if_recursion_tail_stop_do:Nn #3 { #1 }
1084     \_siunitx_number_round_truncate:nnN {#1#2} {#3}
1085   }

```

(End definition for _siunitx_number_round:nnn and others.)

_siunitx_number_round_if_half_p:n A simple test for a valuing being exactly half: we can only test digit-by-digit as there is no limit on the size of the value given.

```

1086 \prg_new_conditional:Npnn \_siunitx_number_round_if_half:n #1 { p }
1087   {
1088     \int_compare:nNnTF { \tl_head:n { #1 0 } } = 5
1089     {
1090       \exp_after:wN \_siunitx_number_round_if_half:N \use_none:n #1 0
1091       \q_recursion_tail \q_recursion_stop
1092     }
1093     { \prg_return_false: }
1094   }
1095 \cs_new:Npn \_siunitx_number_round_if_half:N #1
1096   {
1097     \quark_if_recursion_tail_stop_do:Nn #1
1098     { \prg_return_true: }
1099     \int_compare:nNnTF {#1} = 0
1100     { \_siunitx_number_round_if_half:N }
1101     { \use_i_delimit_by_q_recursion_stop:nw { \prg_return_false: } }
1102   }

```

(End definition for `__siunitx_number_round_if_half_p:n` and `__siunitx_number_round_if_half:N`.)

`__siunitx_number_round_pad:nnn` The case where we are short of digits is easy enough to handle: generate zeros to pad it out.

```

1103 \cs_new:Npn \__siunitx_number_round_pad:nnn #1#2#3
1104 {
1105   {#2}
1106   {
1107     #3
1108     \bool_if:NT \l__siunitx_number_round_pad_bool
1109     { \prg_replicate:nn {#1} { 0 } }
1110   }
1111 }

```

(End definition for `__siunitx_number_round_pad:nnn`.)

`__siunitx_number_round_figures:nnnnnnn` Rounding to a fixed number of significant figures starts by checking that there is no
`__siunitx_number_round_figures_count:nnN` uncertainty, and that the number of figures requested is positive: if not, the result is
`__siunitx_number_round_figures_count:nnnN` always fixed at zero.

```

1112 \cs_new:Npn \__siunitx_number_round_figures:nnnnnnn #1#2#3#4#5#6#7
1113 {
1114   \tl_if_blank:nTF {#5}
1115   {
1116     \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1117     {
1118       \exp_not:n { {#1} {#2} }
1119       \__siunitx_number_round_figures_count:nnN {#3} {#4} #3#4
1120       \q_recursion_tail \q_recursion_stop
1121       \exp_not:n { { } {#6} {#7} }
1122     }
1123     { { } { } { 0 } { } { } { } { 0 } }
1124   }
1125   { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1126 }

```

The first real step is to count up the number of significant figures. The only tricky issue here is dealing with leading zeros.

```

1127 \cs_new:Npn \__siunitx_number_round_figures_count:nnN #1#2#3
1128 {
1129   \quark_if_recursion_tail_stop_do:Nn #3
1130   { { } { } { 0 } { } { } { } { 0 } }
1131   \int_compare:nNnTF {#3} = 0
1132   { \__siunitx_number_round_figures_count:nnN {#1} {#2} }
1133   { \__siunitx_number_round_figures_count:nnnN { 1 } {#1} {#2} }
1134 }
1135 \cs_new:Npn \__siunitx_number_round_figures_count:nnnN #1#2#3#4
1136 {
1137   \quark_if_recursion_tail_stop_do:Nn #4
1138   {
1139     \int_compare:nNnTF {#1} > \l__siunitx_number_round_precision_int
1140     {
1141       \__siunitx_number_round:fnN
1142       { \int_eval:n { #1 - \l__siunitx_number_round_precision_int } }
1143       {#2} {#3}

```



```

1144     }
1145     {
1146         \__siunitx_number_round_pad:nnn
1147         { \l__siunitx_number_round_precision_int - (#1) } {#2} {#3}
1148     }
1149 }
1150 \exp_args:Nf \__siunitx_number_round_figures_count:nnnN
1151 { \int_eval:n { #1 + 1 } } {#2} {#3}
1152 }

```

(End definition for __siunitx_number_round_figures:nnnnnnn, __siunitx_number_round_figures_count:nnN, and __siunitx_number_round_figures_count:nnnN.)

```

\__siunitx_number_round_places:nnnnnnn
\__siunitx_number_round_places_end:nn
\__siunitx_number_round_places_decimal:nn
\__siunitx_number_round_places_integer:nn

```

The first step when rounding to a fixed number of places is to establish if this is in the decimal or integer parts. The two require different calculations for how many digits to drop from the input. The no-op end function here is to allow tidying up in some cases: see the finalisation of rounding.

```

1153 \cs_new:Npn \__siunitx_number_round_places:nnnnnnn #1#2#3#4#5#6#7
1154 {
1155     \tl_if_blank:nTF {#5}
1156     {
1157         \exp_not:n { {#1} {#2} }
1158         \int_compare:nNnTF \l__siunitx_number_round_precision_int > 0
1159         { \__siunitx_number_round_places_decimal:nn }
1160         { \__siunitx_number_round_places_integer:nn }
1161         {#3} {#4}
1162         \__siunitx_number_round_places_end:nn {#6} {#7}
1163     }
1164     { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1165 }
1166 \cs_new:Npn \__siunitx_number_round_places_end:nn #1#2 { { } \exp_not:n { {#1} {#2} } }
1167 \cs_new:Npn \__siunitx_number_round_places_decimal:nn #1#2
1168 {
1169     \int_compare:nNnTF
1170     { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} } > 0
1171     {
1172         \__siunitx_number_round_pad:nnn
1173         { \l__siunitx_number_round_precision_int - 0 \tl_count:n {#2} }
1174         {#1} {#2}
1175     }
1176     {
1177         \__siunitx_number_round:fnn
1178         {
1179             \int_eval:n
1180             { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1181         }
1182         {#1} {#2}
1183     }
1184 }
1185 \cs_new:Npn \__siunitx_number_round_places_integer:nn #1#2
1186 {
1187     \__siunitx_number_round:fnn
1188     {
1189         \int_eval:n

```

```

1190         { 0 \tl_count:n {#2} - \l__siunitx_number_round_precision_int }
1191     }
1192     {#1} {#2}
1193 }

```

(End definition for `__siunitx_number_round_places:nnnnnnn` and others.)

`__siunitx_number_round_uncertainty:nnnnnnn` Rounding to an uncertainty can only happen where the result will have some uncertainty left: otherwise we simply drop the uncertainty entirely. Only S-type uncertainties can be used for rounding.

```

1194 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnnnn #1#2#3#4#5#6#7
1195 {
1196     \bool_lazy_or:nnTF
1197     { \tl_if_blank_p:n {#5} }
1198     { ! \int_compare_p:nNn \l__siunitx_number_round_precision_int > 0 }
1199     { \exp_not:n { {#1} #2 {#3} {#4} { } #6 {#7} } }
1200     {
1201         \str_if_eq:eeTF { \tl_head:n {#5} } { S }
1202         {
1203             \exp_not:n { {#1} {#2} }
1204             \exp_args:Nnno \__siunitx_number_round_uncertainty:nnn
1205             {#3} {#4} { \use_ii:nn #5 }
1206             \exp_not:n { {#6} {#7} }
1207         }
1208         { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#7} } }
1209     }
1210 }

```

Round the uncertainty first: this is needed to get the number of places correct (for the case where the uncertainty rounds up to 1...). Once that is done, it's just a question of working out the digits in the main part.

```

1211 \cs_new:Npn \__siunitx_number_round_uncertainty:nnn #1#2#3
1212 {
1213     \exp_last_unbraced:Nf \__siunitx_number_round_uncertainty:nnnnn
1214     {
1215         \__siunitx_number_round:fnn
1216         { \tl_count:n {#3} - \l__siunitx_number_round_precision_int } { } {#3}
1217     }
1218     {#1} {#2} {#3}
1219 }
1220 \cs_new:Npn \__siunitx_number_round_uncertainty:nnnnn #1#2#3#4#5
1221 {
1222     \tl_if_blank:nTF {#1}
1223     {
1224         \__siunitx_number_round:fnn
1225         { \tl_count:n {#5} - \tl_count:n {#2} } {#3} {#4}
1226         { { S } {#2} }
1227     }
1228     {
1229         \__siunitx_number_round:fnn
1230         { \tl_count:n {#5} - \tl_count:n {#2} + 1 } {#3} {#4}
1231         { { S } { #1 \__siunitx_number_round_truncate_direct:n {#2} } }
1232     }
1233 }

```

(End definition for `__siunitx_number_round_uncertainty:nnnnnnn`, `__siunitx_number_round_uncertainty:nnn`, and `__siunitx_number_round_uncertainty:nnnnn`.)

Simple stripping of the decimal part if zero.

```

\__siunitx_number_zero_decimal:NN
\__siunitx_number_zero_decimal:nnnnnnn
1234 \cs_new_protected:Npn \__siunitx_number_zero_decimal:NN #1#2
1235 {
1236   \bool_if:NT \l__siunitx_number_drop_zero_decimal_bool
1237   {
1238     \tl_set:Nx #2
1239     { \exp_after:wN \__siunitx_number_zero_decimal:nnnnnnn #1 }
1240   }
1241 }
1242 \cs_new:Npn \__siunitx_number_zero_decimal:nnnnnnn #1#2#3#4#5#6#7
1243 {
1244   \exp_not:n { {#1} {#2} {#3} }
1245   \str_if_eq:eeTF
1246   { \exp_not:n {#4} }
1247   { \prg_replicate:nn { \tl_count:n {#4} } { 0 } }
1248   { { } }
1249   { \exp_not:n { {#4} } }
1250   \exp_not:n { {#5} {#6} {#7} }
1251 }

```

(End definition for `__siunitx_number_zero_decimal:NN` and `__siunitx_number_zero_decimal:nnnnnnn`.)

2.5 Formatting parsed numbers

`\l_siunitx_number_output_decimal_tl`

```

1252 \tl_new:N \l_siunitx_number_output_decimal_tl

```

(End definition for `\l_siunitx_number_output_decimal_tl`. This variable is documented on page 22.)

Keys producing tokens in the output.

```

\l_siunitx_number_bracket_negative_bool
\l_siunitx_number_bracket_close_tl
\l_siunitx_number_implicit_plus_bool
\l_siunitx_number_exponent_base_tl
\l_siunitx_number_exponent_product_tl
\l_siunitx_number_group_decimal_bool
\l_siunitx_number_group_integer_bool
\l_siunitx_number_group_minimum_int
\l_siunitx_number_group_separator_tl
\l_siunitx_number_negative_color_tl
\l_siunitx_number_bracket_open_tl
\l_siunitx_number_output_uncert_close_tl
\l_siunitx_number_output_uncert_open_tl
\l_siunitx_number_uncert_separate_bool
\l_siunitx_number_uncert_separator_tl
\l_siunitx_number_tight_bool
\l_siunitx_number_unity_mantissa_bool
\l_siunitx_number_zero_exponent_bool
1253 \keys_define:nn { siunitx }
1254 {
1255   bracket-negative-numbers .bool_set:N =
1256   \l_siunitx_number_bracket_negative_bool ,
1257   exponent-base .tl_set:N =
1258   \l_siunitx_number_exponent_base_tl ,
1259   exponent-product .tl_set:N =
1260   \l_siunitx_number_exponent_product_tl ,
1261   group-digits .choice: ,
1262   group-digits / all .code:n =
1263   {
1264     \bool_set_true:N \l_siunitx_number_group_decimal_bool
1265     \bool_set_true:N \l_siunitx_number_group_integer_bool
1266   } ,
1267   group-digits / decimal .code:n =
1268   {
1269     \bool_set_true:N \l_siunitx_number_group_decimal_bool
1270     \bool_set_false:N \l_siunitx_number_group_integer_bool
1271   } ,
1272   group-digits / integer .code:n =
1273   {

```

```

1274         \bool_set_false:N \l__siunitx_number_group_decimal_bool
1275         \bool_set_true:N \l__siunitx_number_group_integer_bool
1276     } ,
1277     group-digits / none .code:n =
1278     {
1279         \bool_set_false:N \l__siunitx_number_group_decimal_bool
1280         \bool_set_false:N \l__siunitx_number_group_integer_bool
1281     } ,
1282     group-digits .default:n = all ,
1283     group-minimum-digits .int_set:N =
1284         \l__siunitx_number_group_minimum_int ,
1285     group-separator .tl_set:N =
1286         \l__siunitx_number_group_separator_tl ,
1287     negative-color .tl_set:N =
1288         \l__siunitx_number_negative_color_tl ,
1289     number-close-bracket .tl_set:N =
1290         \l__siunitx_number_bracket_close_tl ,
1291     number-open-bracket .tl_set:N =
1292         \l__siunitx_number_bracket_open_tl ,
1293     output-close-uncertainty .tl_set:N =
1294         \l__siunitx_number_output_uncert_close_tl ,
1295     output-decimal-marker .tl_set:N =
1296         \l__siunitx_number_output_decimal_tl ,
1297     output-open-uncertainty .tl_set:N =
1298         \l__siunitx_number_output_uncert_open_tl ,
1299     print-implicit-plus .bool_set:N =
1300         \l__siunitx_number_implicit_plus_bool ,
1301     print-unity-mantissa .bool_set:N =
1302         \l__siunitx_number_unity_mantissa_bool ,
1303     print-zero-exponent .bool_set:N =
1304         \l__siunitx_number_zero_exponent_bool ,
1305     separate-uncertainty .bool_set:N =
1306         \l__siunitx_number_uncert_separate_bool ,
1307     uncertainty-separator .tl_set:N =
1308         \l__siunitx_number_uncert_separator_tl ,
1309     tight-spacing .bool_set:N =
1310         \l__siunitx_number_tight_bool
1311 }
1312 \bool_new:N \l__siunitx_number_group_decimal_bool
1313 \bool_new:N \l__siunitx_number_group_integer_bool

```

(End definition for `\l__siunitx_number_bracket_negative_bool` and others.)

\siunitx_number_format:N

\siunitx_number_format:NN

__siunitx_number_format:Nn

__siunitx_number_format:nnnnnn

__siunitx_number_format_comparator:nn

__siunitx_number_format_sign:n

__siunitx_number_format_sign:N

__siunitx_number_format_sign_color:w

__siunitx_number_format_sign_brackets:w

__siunitx_number_format_integer:nnn

__siunitx_number_format_decimal:nn

__siunitx_number_format_decimal:fn

__siunitx_number_format_digits:nn

__siunitx_number_format_integer_aux:n

__siunitx_number_format_integer_aux_0:n

__siunitx_number_format_integer_aux_1:n

__siunitx_number_format_integer_aux_2:n

__siunitx_number_format_decimal_aux:n

__siunitx_number_format_decimal_loop:NNNN

__siunitx_number_format_integer_first:nnNN

__siunitx_number_format_integer_loop:NNNN

The approach to formatting a single number is to split into the constituent parts. All of the parts are assembled including inserting tabular alignment markers (which may be empty) for each separate unit.

```

1314 \cs_new:Npn \siunitx_number_format:N #1
1315 { \__siunitx_number_format:Nn #1 { } }
1316 \cs_new:Npn \siunitx_number_format:NN #1#2
1317 { \__siunitx_number_format:Nn #1 {#2} }
1318 \cs_new:Npn \__siunitx_number_format:Nn #1#2
1319 {
1320     \tl_if_empty:NF #1
1321     { \exp_after:wN \__siunitx_number_format:nnnnnnn #1 {#2} }

```

```

1322 }
1323 \cs_new:Npn \__siunitx_number_format:nnnnnnn #1#2#3#4#5#6#7#8
1324 {
1325   \__siunitx_number_format_comparator:nn {#1} {#8}
1326   \__siunitx_number_format_sign:n {#2}
1327   \__siunitx_number_format_integer:nnn {#3} {#4} {#7}
1328   \__siunitx_number_format_decimal:nn {#4} {#8}
1329   \__siunitx_number_format_uncertainty:nnn {#5} {#4} {#8}
1330   \__siunitx_number_format_exponent:nnnn {#6} {#7} { #3 . #4 } {#8}
1331   \__siunitx_number_format_end:
1332 }

```

To get the spacing correct this needs to be an ordinary math character.

```

1333 \cs_new:Npn \__siunitx_number_format_comparator:nn #1#2
1334 {
1335   \tl_if_blank:nF {#1}
1336   { \exp_not:n { \mathord {#1} } }
1337   \exp_not:n {#2}
1338 }

```

Formatting signs has to deal with some additional formatting requirements for negative numbers. Both making such numbers a fixed color and bracketing them needs some rearrangement of the order of tokens, which is set up in the main formatting macro by the dedicated do-nothing end function.

```

1339 \cs_new:Npn \__siunitx_number_format_sign:n #1
1340 {
1341   \tl_if_blank:nTF {#1}
1342   {
1343     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1344     { \__siunitx_number_format_sign:N + }
1345   }
1346   {
1347     \str_if_eq:nnTF {#1} { - }
1348     {
1349       \tl_if_empty:NF \l__siunitx_number_negative_color_tl
1350       { \__siunitx_number_format_sign_color:w }
1351       \bool_if:NTF \l__siunitx_number_bracket_negative_bool
1352       { \__siunitx_number_format_sign_brackets:w }
1353       { \__siunitx_number_format_sign:N #1 }
1354     }
1355     { \__siunitx_number_format_sign:N #1 }
1356   }
1357 }
1358 \cs_new:Npn \__siunitx_number_format_sign:N #1
1359 {
1360   \bool_if:NTF \l__siunitx_number_tight_bool
1361   { \exp_not:n { \mathord {#1} } }
1362   { \exp_not:n {#1} }
1363 }
1364 \cs_new:Npn
1365   \__siunitx_number_format_sign_color:w #1 \__siunitx_number_format_end:
1366 {
1367   \exp_not:N \textcolor { \exp_not:V \l__siunitx_number_negative_color_tl }
1368   {
1369     #1

```

```

1370     \__siunitx_number_format_end:
1371   }
1372 }
1373 \cs_new:Npn
1374   \__siunitx_number_format_sign_brackets:w #1 \__siunitx_number_format_end:
1375 {
1376   \exp_not:V \l__siunitx_number_bracket_open_tl
1377   #1
1378   \exp_not:V \l__siunitx_number_bracket_close_tl
1379   \__siunitx_number_format_end:
1380 }

```

Digit formatting leads off with separate functions to allow for a few “up front” items before using a common set of tests for some common cases. The code then splits again as the two types of grouping need different strategies.

```

1381 \cs_new:Npn \__siunitx_number_format_integer:nnn #1#2#3
1382 {
1383   \bool_lazy_all:nF
1384   {
1385     { \str_if_eq_p:nn {#1} { 1 } }
1386     { \tl_if_blank_p:n {#2} }
1387     { ! \str_if_eq_p:nn {#3} { 0 } }
1388     { ! \l__siunitx_number_unity_mantissa_bool }
1389   }
1390   { \__siunitx_number_format_digits:nn { integer } {#1} }
1391 }
1392 \cs_new:Npn \__siunitx_number_format_decimal:nn #1#2
1393 {
1394   \exp_not:n {#2}
1395   \tl_if_blank:nF {#1}
1396   {
1397     \str_if_eq:VnTF \l__siunitx_number_output_decimal_tl { , }
1398     { \exp_not:N \mathord }
1399     { \use:n }
1400     { \exp_not:V \l__siunitx_number_output_decimal_tl }
1401   }
1402   \exp_not:n {#2}
1403   \__siunitx_number_format_digits:nn { decimal } {#1}
1404 }
1405 \cs_generate_variant:Nn \__siunitx_number_format_decimal:nn { f }
1406 \cs_new:Npn \__siunitx_number_format_digits:nn #1#2
1407 {
1408   \bool_if:cTF { l__siunitx_number_group_ #1 _ bool }
1409   {
1410     \int_compare:nNnTF
1411     { \tl_count:n {#2} } < \l__siunitx_number_group_minimum_int
1412     { \exp_not:n {#2} }
1413     { \use:c { __siunitx_number_format_ #1 _aux:n } {#2} }
1414   }
1415   { \exp_not:n {#2} }
1416 }

```

For integers, we need to know how many digits there are to allow for the correct insertion of separators. That is done using a two-part set up such that there is no separator on the first pass.

```

1417 \cs_new:Npn \__siunitx_number_format_integer_aux:n #1
1418 {
1419   \use:c
1420   {
1421     __siunitx_number_format_integer_aux_
1422     \int_eval:n { \int_mod:nn { \tl_count:n {#1} } { 3 } }
1423     :n
1424   } {#1}
1425 }
1426 \cs_new:cpn { __siunitx_number_format_integer_aux_0:n } #1
1427 { \__siunitx_number_format_integer_first:nnNN #1 \q_nil }
1428 \cs_new:cpn { __siunitx_number_format_integer_aux_1:n } #1
1429 { \__siunitx_number_format_integer_first:nnNN { } { } #1 \q_nil }
1430 \cs_new:cpn { __siunitx_number_format_integer_aux_2:n } #1
1431 { \__siunitx_number_format_integer_first:nnNN { } #1 \q_nil }
1432 \cs_new:Npn \__siunitx_number_format_integer_first:nnNN #1#2#3#4
1433 {
1434   \exp_not:n {#1#2#3}
1435   \quark_if_nil:NF #4
1436   { \__siunitx_number_format_integer_loop:NNNN #4 }
1437 }
1438 \cs_new:Npn \__siunitx_number_format_integer_loop:NNNN #1#2#3#4
1439 {
1440   \str_if_eq:VnTF \l__siunitx_number_group_separator_tl { , }
1441   { \exp_not:N \mathord }
1442   { \use:n }
1443   { \exp_not:V \l__siunitx_number_group_separator_tl }
1444   \exp_not:n {#1#2#3}
1445   \quark_if_nil:NF #4
1446   { \__siunitx_number_format_integer_loop:NNNN #4 }
1447 }

```

For decimals, no need to do any counting, just loop using enough markers to find the end of the list. By passing the decimal marker, it is possible not to have to use a check on the content of the rest of the number. The `\use_none:n(n)` mop up the remaining `\q_nil` tokens.

```

1448 \cs_new:Npn \__siunitx_number_format_decimal_aux:n #1
1449 {
1450   \__siunitx_number_format_decimal_loop:NNNN \c_empty_tl
1451   #1 \q_nil \q_nil \q_nil
1452 }
1453 \cs_new:Npn \__siunitx_number_format_decimal_loop:NNNN #1#2#3#4
1454 {
1455   \quark_if_nil:NF #2
1456   {
1457     \exp_not:V #1
1458     \exp_not:n {#2}
1459     \quark_if_nil:NTF #3
1460     { \use_none:n }
1461     {
1462       \exp_not:n {#3}
1463       \quark_if_nil:NTF #4
1464       { \use_none:nn }
1465       {

```

```

1466         \exp_not:n {#4}
1467         \__siunitx_number_format_decimal_loop:NNNN
1468         \l__siunitx_number_group_separator_tl
1469     }
1470 }
1471 }
1472 }

```

Uncertainties which are directly attached are easy to deal with. For those that are separated, the first step is to find if they are entirely contained within the decimal part, and to pad if they are. For the case where the boundary is crossed to the integer part, the correct number of digit tokens need to be removed from the start of the uncertainty and the split result sent to the appropriate auxiliaries.

```

1473 \cs_new:Npn \__siunitx_number_format_uncertainty:nnn #1#2#3
1474 {
1475     \tl_if_blank:nTF {#1}
1476     { \__siunitx_number_format_uncertainty_unaligned:n {#3} }
1477     {
1478         \use:c { \__siunitx_number_format_uncertainty_ \tl_head:n {#1} :nnnw }
1479         {#2} {#3} #1
1480     }
1481 }
1482 \cs_new:Npn \__siunitx_number_format_uncertainty_unaligned:n #1
1483 { \exp_not:n { #1 #1 #1 #1 } }
1484 \cs_new:Npn \__siunitx_number_format_uncertainty_S:nnnw #1#2#3#4
1485 {
1486     \bool_if:NTF \l__siunitx_number_uncert_separate_bool
1487     {
1488         \exp_not:n {#2}
1489         \__siunitx_number_format_sign:N \pm
1490         \exp_not:n {#2}
1491         \exp_args:Nf \__siunitx_number_format_uncertainty_S_aux:nnn
1492         { \int_eval:n { \tl_count:n {#4} - \tl_count:n {#1} } }
1493         {#4} {#2}
1494     }
1495     {
1496         \exp_not:V \l__siunitx_number_uncert_separator_tl
1497         \exp_not:V \l__siunitx_number_output_uncert_open_tl
1498         \exp_not:n {#4}
1499         \exp_not:V \l__siunitx_number_output_uncert_close_tl
1500         \__siunitx_number_format_uncertainty_unaligned:n {#2}
1501     }
1502 }
1503 \cs_new:Npn \__siunitx_number_format_uncertainty_S_aux:nnn #1#2#3
1504 {
1505     \int_compare:nNnTF {#1} > 0
1506     {
1507         \__siunitx_number_format_uncertainty_S_aux:fnnw
1508         { \int_eval:n { #1 - 1 } }
1509         {#3}
1510         { }
1511         #2 \q_nil
1512     }
1513     {

```



```

1514         0
1515         \_siunitx_number_format_decimal:fn
1516         {
1517             \prg_replicate:nn { \int_abs:n {#1} } { 0 }
1518             #2
1519         }
1520         {#3}
1521     }
1522 }
1523 \cs_new:Npn \_siunitx_number_format_uncertainty_S_aux:nnnw #1#2#3#4
1524 {
1525     \quark_if_nil:NF #4
1526     {
1527         \int_compare:nNnTF {#1} = 0
1528         { \_siunitx_number_format_uncertainty_S_aux:nnw {#3#4} {#2} }
1529         {
1530             \_siunitx_number_format_uncertainty_S_aux:fnnw
1531             { \int_eval:n { #1 - 1 } }
1532             {#2}
1533             {#3#4}
1534         }
1535     }
1536 }
1537 \cs_generate_variant:Nn \_siunitx_number_format_uncertainty_S_aux:nnnw { f }
1538 \cs_new:Npn \_siunitx_number_format_uncertainty_S_aux:nnw #1#2#3 \q_nil
1539 {
1540     \_siunitx_number_format_digits:nn { integer } {#1}
1541     \_siunitx_number_format_decimal:nn {#3} {#2}
1542 }

```

Setting the exponent part requires some information about the mantissa: was it there or not. This means that whilst only the sign and value for the exponent are typeset here, there is a need to also have access to the combined mantissa part (with a decimal marker). The rest of the work is about picking up the various options and getting the combinations right. For signs, the auxiliary from the main sign routine can be used, but not the main function: negative exponents don't have special handling.

```

1543 \cs_new:Npn \_siunitx_number_format_exponent:nnnn #1#2#3#4
1544 {
1545     \exp_not:n {#4}
1546     \bool_lazy_or:nnTF
1547     { \l__siunitx_number_zero_exponent_bool }
1548     { ! \str_if_eq_p:nn {#2} { 0 } }
1549     {
1550         \bool_lazy_and:nnTF
1551         { \str_if_eq_p:nn {#3} { 1. } }
1552         { ! \l__siunitx_number_unity_mantissa_bool }
1553         { \exp_not:n {#4} }
1554         {
1555             \bool_if:NTF \l__siunitx_number_tight_bool
1556             { \exp_not:N \mathord }
1557             { \use:n }
1558             { \exp_not:V \l__siunitx_number_exponent_product_tl }
1559             \exp_not:n {#4}
1560         }
1561     }

```

```

1561 \exp_not:V \l__siunitx_number_exponent_base_tl
1562 ^
1563 {
1564   \tl_if_blank:nTF {#1}
1565   {
1566     \bool_if:NT \l__siunitx_number_implicit_plus_bool
1567     { \__siunitx_number_format_sign:N + }
1568   }
1569   { \__siunitx_number_format_sign:N #1 }
1570   \__siunitx_number_format_digits:nn { integer } {#2}
1571 }
1572 }
1573 { \exp_not:n {#4} }
1574 }

```

A do-nothing marker used to allow shuffling of the output and so expandable operations for formatting.

```
1575 \cs_new:Npn \__siunitx_number_format_end: { }
```

(End definition for `\siunitx_number_format:N` and others. These functions are documented on page 22.)

2.6 Miscellaneous tools

`\l__siunitx_number_valid_tl` The list of valid tokens.

```
1576 \tl_new:N \l__siunitx_number_valid_tl
```

(End definition for `\l__siunitx_number_valid_tl`.)

`\siunitx_if_number:nTF` Test if an entire number is valid: this means parsing the number but not returning anything.

```

1577 \prg_new_protected_conditional:Npnn \siunitx_if_number:n #1
1578 { T , F , TF }
1579 {
1580   \group_begin:
1581     \bool_set_true:N \l__siunitx_number_validate_bool
1582     \bool_set_true:N \l__siunitx_number_parse_bool
1583     \siunitx_number_parse:nN {#1} \l__siunitx_number_parsed_tl
1584     \tl_if_empty:NTF \l__siunitx_number_parsed_tl
1585     {
1586       \group_end:
1587       \prg_return_false:
1588     }
1589     {
1590       \group_end:
1591       \prg_return_true:
1592     }
1593 }

```

(End definition for `\siunitx_if_number:nTF`. This function is documented on page 22.)

`\siunitx_if_number_token_p:N` A simple conditional to answer the question of whether a specific token is possibly valid in a number.

`\siunitx_if_number_token:NTF`

`__siunitx_number_if_number_token_auxi:NN`

`__siunitx_number_if_number_token_auxii:NN`

`__siunitx_number_if_number_token_auxiii:NN`

```

1594 \prg_new_conditional:Npnn \siunitx_if_number_token:N #1
1595 { p , T , F , TF }

```

```

1596 {
1597   \__siunitx_number_number_token_auxi:NN #1
1598   \l_siunitx_number_input_decimal_tl
1599   \l__siunitx_number_input_uncert_close_tl
1600   \l_siunitx_number_input_comparator_tl
1601   \l__siunitx_number_input_digit_tl
1602   \l_siunitx_number_input_exponent_tl
1603   \l__siunitx_number_input_ignore_tl
1604   \l_siunitx_number_input_uncert_open_tl
1605   \l__siunitx_number_input_sign_tl
1606   \l_siunitx_number_input_uncert_sign_tl
1607   \q_recursion_tail
1608   \q_recursion_stop
1609 }
1610 \cs_new:Npn \__siunitx_number_number_token_auxi:NN #1#2
1611 {
1612   \quark_if_recursion_tail_stop_do:Nn #2 { \prg_return_false: }
1613   \__siunitx_number_number_token_auxii:NN #1 #2
1614   \__siunitx_number_number_token_auxi:NN #1
1615 }
1616 \cs_new:Npn \__siunitx_number_number_token_auxii:NN #1#2
1617 {
1618   \exp_after:wN \__siunitx_number_number_token_auxiii:NN \exp_after:wN #1
1619   #2 \q_recursion_tail \q_recursion_stop
1620 }
1621 \cs_new:Npn \__siunitx_number_number_token_auxiii:NN #1#2
1622 {
1623   \quark_if_recursion_tail_stop:N #2
1624   \str_if_eq:nnT {#1} {#2}
1625   {
1626     \use_i_delimit_by_q_recursion_stop:nw
1627     {
1628       \use_i_delimit_by_q_recursion_stop:nw
1629       { \prg_return_true: }
1630     }
1631   }
1632   \__siunitx_number_number_token_auxiii:NN #1
1633 }

```

(End definition for `\siunitx_if_number_token:NTF` and others. This function is documented on page [22](#).)

2.7 Messages

```

1634 \msg_new:nnnn { siunitx } { invalid-number }
1635 { Invalid-number~'#1'. }
1636 {
1637   The~input~'#1'~could~not~be~parsed~as~a~number~following~the~
1638   format~defined~in~module~documentation.
1639 }

```

2.8 Standard settings for module options

Some of these follow naturally from the point of definition (*e.g.* boolean variables are always `false` to begin with), but for clarity everything is set here.

```

1640 \keys_set:nn { siunitx }
1641 {
1642   bracket-negative-numbers = false ,
1643   drop-exponent            = false ,
1644   drop-uncertainty         = false ,
1645   drop-zero-decimal        = false ,
1646   evaluate-expression      = false ,
1647   exponent-base            = 10 ,
1648   exponent-mode            = input ,
1649   exponent-product         = \times ,
1650   expression               = #1 ,
1651   fixed-exponent           = 0 ,
1652   group-digits             = all ,
1653   group-minimum-digits     = 4 ,
1654   group-separator          = \, , % (
1655   input-close-uncertainty  = ) ,
1656   input-comparators        = { <=>\approx\ge\geq\gg\le\leq\ll\sim } ,
1657   input-decimal-markers    = { ., } ,
1658   input-digits             = 0123456789 ,
1659   input-exponent-markers   = dDeE ,
1660   input-ignore             = \, ,
1661   input-open-uncertainty   = ( , % )
1662   input-signs              = +-\mp\pm ,
1663   input-uncertainty-signs  = \pm ,
1664   minimum-decimal-digits   = 0 ,
1665   minimum-integer-digits   = 0 ,
1666   negative-color           = , % (
1667   number-close-bracket     = ) ,
1668   number-open-bracket      = ( , % ) (
1669   output-close-uncertainty  = ) ,
1670   output-decimal-marker    = . ,
1671   output-open-uncertainty  = ( , % )
1672   parse-numbers            = true ,
1673   print-implicit-plus      = false ,
1674   print-unity-mantissa     = true ,
1675   print-zero-exponent      = false ,
1676   round-half               = up ,
1677   round-minimum            = 0 ,
1678   round-mode               = none ,
1679   round-pad                = true ,
1680   round-precision          = 2 ,
1681   separate-uncertainty     = false ,
1682   track-explicit-plus      = false ,
1683   uncertainty-separator    = ,
1684   tight-spacing            = false
1685 }
1686 \end{package}

```

Part V

siunitx-print – Printing material with font control

1 Printing quantities

This submodule is focussed on providing controlled printing for numbers and units. Key to this is control of font: conventions for printing quantities mean that the exact nature of the output is important. At the same time, this module provides flexibility for the user in terms of which aspects of the font are responsive to the surrounding general text. Printing material may also take place in text or math mode.

The printing routines assume that normal L^AT_EX 2_ε font selection commands are available, in particular `\bfseries`, `\mathrm`, `\mathversion`, `\fontfamily`, `\fontseries` and `\fontshape`, `\familydefault`, `\seriesdefault`, `\shapedefault` and `\selectfont`. It also requires the standard L^AT_EX 2_ε kernel commands `\ensuremath`, `\textsubscript` and `\textsuperscript` for printing in text mode. The following packages are also required to provide the functionality detailed.

- `color`: support for color using `\textcolor`
- `textcomp`: `\textminus` and `\textpm` for printing in text mode
- `amstext`: the `\text` command for printing in text mode

<code>\siunitx_print:nn</code>	<code>\siunitx_print:nn {<type>} {<material>}</code>
<code>\siunitx_print:nV</code>	

Prints the *<material>* according to the prevailing settings for the submodule as applicable to the *<type>* of content: the latter should be either **number** or **unit**. The *<material>* should comprise normal L^AT_EX mark-up for numbers or units. In particular, units will typically use `\mathrm` to indicate material to be printed in the current upright roman font, and `^` and `_` will typically be used to indicate super- and subscripts, respectively. These elements will be correctly handled when printing for example using `\mathsf` in math mode, or using only text fonts.

<code>\siunitx_print_match:n</code>	<code>\siunitx_print_match:n {<material>}</code>
<code>\siunitx_print_math:n</code>	<code>\siunitx_print_math:n {<material>}</code>
<code>\siunitx_print_text:n</code>	<code>\siunitx_print_text:n {<material>}</code>

Prints the *<material>* as described for `\siunitx_print:nn` but with a fixed text or math mode output. The printing does *not* set color (which is managed on a **unit/number** basis), but otherwise sets the font as described above. The **match** function uses either the prevailing math or text mode.

1.1 Key-value options

The options defined by this submodule are available within the l3keys `siunitx` tree.

<code>color</code>	<code>color = <color></code>
--------------------	------------------------------------

Color to apply to printed output: the latter should be a named color defined for use with `\textcolor`. The standard setting is empty (no color).

<u>mode</u>	<code>mode = match math text</code> Selects which mode (math or text) the output is printed in: a choice from the options <code>match</code> , <code>math</code> or <code>text</code> . The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The <code>math</code> and <code>text</code> options choose the relevant T _E X mode for printing. The standard setting is <code>math</code> .
<u>number-color</u>	<code>number-color = <color></code> Color to apply to numbers in output: the latter should be a named color defined for use with <code>\textcolor</code> . The standard setting is empty (no color).
<u>number-mode</u>	<code>number-mode = match math text</code> Selects which mode (math or text) the numbers are printed in: a choice from the options <code>match</code> , <code>math</code> or <code>text</code> . The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The <code>math</code> and <code>text</code> options choose the relevant T _E X mode for printing. The standard setting is <code>math</code> .
<u>propagate-math-font</u>	<code>propagate-math-font = true false</code> Switch to determine if the currently-active math font is applied within printed output. This is relevant only when <code>\siunitx_print:nn</code> is called from within math mode: in text mode there is not active math font. When not active, math mode material will be typeset using standard math mode fonts without any changes being made to the supplied argument. The standard setting is <code>false</code> .
<u>reset-math-version</u>	<code>reset-math-version = true false</code> Switch to determine whether the active <code>\mathversion</code> is reset to <code>normal</code> when printing in math mode. Note that math version is typically used to select <code>\boldmath</code> , though it is also be used by <i>e.g.</i> <code>sansmath</code> . The standard setting is <code>true</code> .
<u>reset-text-family</u>	<code>reset-text-family = true false</code> Switch to determine whether the active text family is reset to <code>\rmfamily</code> when printing in text mode. The standard setting is <code>true</code> .
<u>reset-text-series</u>	<code>reset-text-series = true false</code> Switch to determine whether the active text series is reset to <code>\mdseries</code> when printing in text mode. The standard setting is <code>true</code> .
<u>reset-text-shape</u>	<code>reset-text-shape = true false</code> Switch to determine whether the active text shape is reset to <code>\upshape</code> when printing in text mode. The standard setting is <code>true</code> .
<u>text-family-to-math</u>	<code>text-family-to-math = true false</code> Switch to determine if the family of the current text font should be applied (where possible) to printing in math mode. The standard setting is <code>false</code> .

<hr/> <hr/> text-weight-to-math	<p><code>text-weight-to-math = true false</code></p> <p>Switch to determine if the weight of the current text font should be applied (where possible) to printing in math mode. This is achieved by setting the <code>\mathversion</code>, and so will override <code>reset-math-version</code>. The mappings between text and math weight are stored in <code>\l_siunitx_print_series_prop</code>. The standard setting is <code>false</code>.</p>
<hr/> <hr/> unit-color	<p><code>unit-color = <color></code></p> <p>Color to apply to units in output: the latter should be a named color defined for use with <code>\textcolor</code>. The standard setting is empty (no color).</p>
<hr/> <hr/> unit-mode	<p><code>unit-mode = match math text</code></p> <p>Selects which mode (math or text) units are printed in: a choice from the options <code>match</code>, <code>math</code> or <code>text</code>. The option <code>match</code> matches the mode prevailing at the point <code>\siunitx_print:nn</code> is called. The <code>math</code> and <code>text</code> options choose the relevant T_EX mode for printing. The standard setting is <code>math</code>.</p>
<hr/> <hr/> weight-version-mapping	<p><code>weight-version-mapping / <weight> = <version></code></p> <p>Defines how <code>siunitx</code> maps from text font weight to math font version. The pre-defined weights are those used as-standard by <code>autoinst</code>:</p> <ul style="list-style-type: none"> • <code>ul</code> • <code>el</code> • <code>l</code> • <code>sl</code> • <code>m</code> • <code>sb</code> • <code>b</code> • <code>eb</code> • <code>ub</code>

As standard, the `m` weight maps to `normal` math version whilst all of the `b` weights map to `bold` and all of the `l` weights map to `light`.

2 siunitx-print implementation

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 <@@=siunitx_print>
```

2.1 Initial set up

The printing routines depend on `amstext` for text mode working.

```

3 \RequirePackage { amstext }
   Color support is always required.
4 \RequirePackage { color }
   For a sensible \textminus we load textcomp if fontspec is not in use.
5 \AtBeginDocument
6 {
7   \@ifpackageloaded { fontspec }
8     { }
9     { \RequirePackage { textcomp } }
10  }
```

`\tl_replace_all:N\N` Required variants.

```
11 \cs_generate_variant:Nn \tl_replace_all:Nnn { NV }
```

(End definition for `\tl_replace_all:N\N`. This function is documented on page ??.)

`\l_siunitx_print_tmp_box` Scratch space.

```

\l_siunitx_print_tmp_tl
12 \box_new:N \l_siunitx_print_tmp_box
13 \tl_new:N \l_siunitx_print_tmp_tl
```

(End definition for `\l_siunitx_print_tmp_box` and `\l_siunitx_print_tmp_tl`.)

`\document`

In order to test math fonts, we need information about the `\fam` used by the various options. This is run as a hook onto `\document`, rather than using `\AtBeginDocument` as it has to come after anything that `fontspec` does (nasty errors arise otherwise). As this is a true one-off, we avoid wasting a box.

```

14 \tl_put_right:Nn \document
15 {
16   \__siunitx_print_store_fam:n { rm }
17   \__siunitx_print_store_fam:n { sf }
18   \__siunitx_print_store_fam:n { tt }
19   \ignorespaces
20 }
21 \cs_new_protected:Npn \__siunitx_print_store_fam:n #1
22 {
23   \group_begin:
24     \hbox_set:Nn \l_siunitx_print_tmp_box
25       {
26         \ensuremath
27           {
28             \use:c { math #1 }
29             { \int_const:cn { c__siunitx_print_math #1 _int } { \fam } }
30           }
31       }
32   \group_end:
33 }
```

(End definition for `\document` and others. This function is documented on page ??.)

2.2 Printing routines

Options which apply to the main formatting routine, and so are not tied to either symbolic or literal input.

```

\l_siunitx_print_number_color_tl
\l_siunitx_print_number_mode_tl
\l_siunitx_print_unit_color_tl
\l_siunitx_print_unit_mode_tl
\l_siunitx_print_math_font_bool
\l_siunitx_print_math_version_bool
\l_siunitx_print_math_family_bool
\l_siunitx_print_math_weight_bool
\l_siunitx_print_text_family_tl
\l_siunitx_print_text_series_tl
\l_siunitx_print_text_shape_tl

34 \tl_new:N \l__siunitx_print_number_mode_tl
35 \tl_new:N \l__siunitx_print_unit_mode_tl
36 \keys_define:nn { siunitx }
37 {
38   color .meta:n =
39     { number-color = #1 , unit-color = #1 } ,
40   mode .meta:n =
41     { number-mode = #1 , unit-mode = #1 } ,
42   number-color .tl_set:N =
43     \l__siunitx_print_number_color_tl ,
44   number-mode .choices:nn =
45     { match , math , text }
46     {
47       \tl_set_eq:NN
48         \l__siunitx_print_number_mode_tl \l_keys_choice_tl
49     } ,
50   propagate-math-font .bool_set:N =
51     \l_siunitx_print_math_font_bool ,
52   reset-math-version .bool_set:N =
53     \l_siunitx_print_math_version_bool ,
54   reset-text-family .bool_set:N =
55     \l_siunitx_print_text_family_bool ,
56   reset-text-series .bool_set:N =
57     \l_siunitx_print_text_series_bool ,
58   reset-text-shape .bool_set:N =
59     \l_siunitx_print_text_shape_bool ,
60   text-family-to-math .bool_set:N =
61     \l_siunitx_print_math_family_bool ,
62   text-weight-to-math .bool_set:N =
63     \l_siunitx_print_math_weight_bool ,
64   unit-color .tl_set:N =
65     \l_siunitx_print_unit_color_tl ,
66   unit-mode .choices:nn =
67     { match , math , text }
68     {
69       \tl_set_eq:NN
70         \l__siunitx_print_unit_mode_tl \l_keys_choice_tl
71     }
72 }

```

(End definition for `\l_siunitx_print_number_color_tl` and others.)

```

\l_siunitx_print_version_ul_tl
\l_siunitx_print_version_el_tl
\l_siunitx_print_version_l_tl
\l_siunitx_print_version_sl_tl
\l_siunitx_print_version_m_tl
\l_siunitx_print_version_sb_tl
\l_siunitx_print_version_b_tl
\l_siunitx_print_version_eb_tl
\l_siunitx_print_version_ub_tl

73 % One set of \enquoef{focussed} options.
74 \keys_define:nn { siunitx / weight-version-mapping }
75 {
76   ul . tl_set:N = \l_siunitx_print_version_ul_tl ,
77   el . tl_set:N = \l_siunitx_print_version_el_tl ,
78   l . tl_set:N = \l_siunitx_print_version_l_tl ,
79   sl . tl_set:N = \l_siunitx_print_version_sl_tl ,

```

```

80   m . tl_set:N = \l__siunitx_print_version_m_tl ,
81   sb . tl_set:N = \l__siunitx_print_version_sb_tl ,
82   b . tl_set:N = \l__siunitx_print_version_b_tl ,
83   eb . tl_set:N = \l__siunitx_print_version_eb_tl ,
84   ub . tl_set:N = \l__siunitx_print_version_ub_tl
85 }

```

(End definition for `\l__siunitx_print_version_ul_tl` and others.)

`\siunitx_print:nn` The main printing function doesn't actually need to do very much: just set the color and
`\siunitx_print:nV` select the correct sub-function.

```

86 \cs_new_protected:Npn \siunitx_print:nn #1#2
87 {
88   \tl_if_empty:cTF { l__siunitx_print_ #1 _color_tl }
89   { \use:n }
90   { \exp_args:Nv \textcolor { l__siunitx_print_ #1 _color_tl } }
91   {
92     \use:c
93     {
94       siunitx_print_
95       \tl_use:c { l__siunitx_print_ #1 _mode_tl } :n
96     }
97     {#2}
98   }
99 }
100 \cs_generate_variant:Nn \siunitx_print:nn { nV }

```

(End definition for `\siunitx_print:nn`. This function is documented on page 65.)

`\siunitx_print_match:n` When the *output* mode should match the input, a simple selection of route can be made.

```

101 \cs_new_protected:Npn \siunitx_print_match:n #1
102 {
103   \mode_if_math:TF
104   { \siunitx_print_math:n {#1} }
105   { \siunitx_print_text:n {#1} }
106 }

```

(End definition for `\siunitx_print_match:n`. This function is documented on page 65.)

`__siunitx_print_replace_font:N` A simple auxiliary for “zapping” the unit font.

```

107 \cs_new_protected:Npn \__siunitx_print_replace_font:N #1
108 {
109   \tl_if_empty:NF \l_siunitx_unit_font_tl
110   {
111     \tl_replace_all:NVn #1
112     \l_siunitx_unit_font_tl
113     { \use:n }
114   }
115 }

```

(End definition for `__siunitx_print_replace_font:N`.)

`\c_siunitx_print_weight_uc_tl` Font widths where the *m* for weight is omitted.

```

116 \clist_map_inline:nn { uc , ec , c , sc , sx , x , ex , ux }
117 { \tl_const:cn { c__siunitx_print_weight_ #1 _tl } { m } }

```

`\c_siunitx_print_weight_ecl_tl`
`\c_siunitx_print_weight_c_tl`
`\c_siunitx_print_weight_sc_tl`
`\c_siunitx_print_weight_sx_tl`
`\c_siunitx_print_weight_x_tl`
`\c_siunitx_print_weight_ex_tl`
`\c_siunitx_print_weight_ux_tl`

(End definition for `\c__siunitx_print_weight_uc_tl` and others.)

`\c__siunitx_print_weight_l_tl` Font widths with one letter.

```
\c__siunitx_print_weight_m_tl 118 \clist_map_inline:nn { l , m , b }
\c__siunitx_print_weight_b_tl 119 { \tl_const:cn { c__siunitx_print_weight_ #1 _tl } { #1 } }
```

(End definition for `\c__siunitx_print_weight_l_tl`, `\c__siunitx_print_weight_m_tl`, and `\c__siunitx_print_weight_b_tl`.)

`\siunitx_print_math:n`

The first step in setting in math mode is to check on the math version. The starting point is the question of whether text series needs to propagate to math mode: if so, check on the mapping, otherwise check on the current math version.

```
\_siunitx_print_extract_series:Nw 120 \cs_new_protected:Npn \siunitx_print_math:n #1
\_siunitx_print_convert_series:n 121 {
\_siunitx_print_math_version:nn 122 \bool_if:NTF \l__siunitx_print_math_weight_bool
\_siunitx_print_math_version:Vn 123 {
\_siunitx_print_math_auxi:n 124 \tl_set:Nx \l__siunitx_print_tmp_tl
\_siunitx_print_math_auxii:n 125 { \exp_after:wN \_siunitx_print_extract_series:Nw \f@series ? \q_stop }
\_siunitx_print_math_auxiii:n 126 \tl_if_empty:NTF \l__siunitx_print_tmp_tl
\_siunitx_print_math_auxiv:n 127 { \_siunitx_print_math_auxi:n {#1} }
\_siunitx_print_math_auxv:n 128 { \_siunitx_print_math_version:Vn \l__siunitx_print_tmp_tl {#1} }
\_siunitx_print_math_aux:Nn 129 }
\_siunitx_print_math_aux:cn 130 { \_siunitx_print_math_auxi:n {#1} }
\_siunitx_print_math_sub:n 131 }
\_siunitx_print_math_super:n
\_siunitx_print_math_script:n
\_siunitx_print_math_text:n
```

Look up the math version from the text series. The weight is omitted if it is `m` plus there are either one or two letters, so we have a little work to do. To keep things fast, we use a hash table based lookup rather than a sequence or property list.

```
132 \cs_new:Npn \_siunitx_print_extract_series:Nw #1#2 ? #3 \q_stop
133 {
134 \cs_if_exist:cTF { c__siunitx_print_weight_ #1#2 _tl }
135 { \_siunitx_print_convert_series:n { m } }
136 {
137 \cs_if_exist:cTF { c__siunitx_print_weight_ #1 _tl }
138 { \_siunitx_print_convert_series:n {#1} }
139 { \_siunitx_print_convert_series:n {#1#2} }
140 }
141 }
142 \cs_new:Npn \_siunitx_print_convert_series:n #1
143 { \tl_use:c { l__siunitx_print_version_ #1 _tl } }
144 \cs_new_protected:Npn \_siunitx_print_math_auxi:n #1
145 {
146 \bool_if:NTF \l__siunitx_print_math_version_bool
147 { \_siunitx_print_math_version:nn { normal } {#1} }
148 { \_siunitx_print_math_auxii:n {#1} }
149 }
```

Any setting which changes the math version can only be set from text mode (as it applies at the level of a formula). As such, the first test is to see if that needs to be to check if the math version has to be set: if so, switch to text mode, sort it out and switch back. That of course means that in such cases, line breaking will not be possible.

```
150 \cs_new_protected:Npn \_siunitx_print_math_version:nn #1#2
151 {
152 \str_if_eq:VnTF \math@version { #1 }
```

```

153 { \__siunitx_print_math_auxii:n {#2} }
154 {
155   \mode_if_math:TF
156   { \text }
157   { \use:n }
158   {
159     \mathversion {#1}
160     \__siunitx_print_math_auxii:n {#2}
161   }
162 }
163 }
164 \cs_generate_variant:Nn \__siunitx_print_math_version:nn { V }

```

At this point, force math mode then start dealing with setting math font based on text family. If the text family is roman, life is slightly different to if it is sanserif or monospaced. In all cases, the outcomes can be handled using the same routines as for normal math mode treatment. The test here is on a string basis as `\f@family` and the `\...default` commands have different `\long` status.

```

165 \cs_new_protected:Npn \__siunitx_print_math_auxii:n #1
166 { \ensuremath { \__siunitx_print_math_auxiii:n {#1} } }
167 \cs_new_protected:Npn \__siunitx_print_math_auxiii:n #1
168 {
169   \bool_if:NTF \l__siunitx_print_math_family_bool
170   {
171     \str_case_e:nnF { \f@family }
172     {
173       { \rmdefault } { \__siunitx_print_math_auxv:n }
174       { \sfdefault } { \__siunitx_print_math_aux:Nn \mathsf }
175       { \ttdefault } { \__siunitx_print_math_aux:Nn \mathtt }
176     }
177     { \__siunitx_print_math_auxiv:n }
178   }
179   { \__siunitx_print_math_auxiv:n }
180   {#1}
181 }

```

Now we deal with the font selection in math mode. There are two possible cases. First, we are retaining the current math font, and the active one is `\mathsf` or `\mathtt`: that needs to be applied to the argument. Alternatively, if the current font is not retained, ensure that normal math mode rules are active. The parts here are split up to allow reuse when picking up the text family.

```

182 \cs_new_protected:Npn \__siunitx_print_math_auxiv:n #1
183 {
184   \bool_if:NTF \l__siunitx_print_math_font_bool
185   {
186     \int_case:nnF \fam
187     {
188       \c__siunitx_print_mathsf_int { \__siunitx_print_math_aux:Nn \mathsf }
189       \c__siunitx_print_mathtt_int { \__siunitx_print_math_aux:Nn \mathtt }
190     }
191     { \use:n }
192   }
193   { \__siunitx_print_math_auxv:n }
194   {#1}

```

```

195 }
196 \cs_new_protected:Npn \__siunitx_print_math_auxv:n #1
197 {
198   \bool_lazy_or:nnTF
199     { \int_compare_p:nNn \fam = { -1 } }
200     { \int_compare_p:nNn \fam = \c__siunitx_print_mathrm_int }
201     { \use:n }
202     { \mathrm }
203     {#1}
204 }

```

Search-and-replace fun: deal with any font commands in the argument and also inside sub/superscripts.

```

205 \cs_new_protected:Npx \__siunitx_print_math_aux:Nn #1#2
206 {
207   \group_begin:
208     \tl_set:Nn \exp_not:N \l__siunitx_print_tmp_tl {#2}
209     \__siunitx_print_replace_font:N \exp_not:N \l__siunitx_print_tmp_tl
210     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
211       { \char_generate:nn { '\_ } { 8 } }
212       { \exp_not:N \__siunitx_print_math_sub:n }
213     \tl_replace_all:Nnn \exp_not:N \l__siunitx_print_tmp_tl
214       { ^ }
215       { \exp_not:N \__siunitx_print_math_super:n }
216     #1 { \exp_not:N \tl_use:N \exp_not:N \l__siunitx_print_tmp_tl }
217   \group_end:
218 }
219 \cs_generate_variant:Nn \__siunitx_print_math_aux:Nn { c }
220 \cs_new_protected:Npx \__siunitx_print_math_sub:n #1
221 {
222   \char_generate:nn { '\_ } { 8 }
223   { \exp_not:N \__siunitx_print_math_script:n {#1} }
224 }
225 \cs_new_protected:Npn \__siunitx_print_math_super:n #1
226 { ^ { \__siunitx_print_math_script:n {#1} } }
227 \cs_new_protected:Npn \__siunitx_print_math_script:n #1
228 {
229   \group_begin:
230     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
231     \__siunitx_print_replace_font:N \l__siunitx_print_tmp_tl
232     \tl_use:N \l__siunitx_print_tmp_tl
233   \group_end:
234 }

```

For tex4ht, we need to have category code 12 ^ tokens in math mode. We handle that by intercepting at the first auxiliary that makes sense.

```

235 \AtBeginDocument
236 {
237   \ifpackageloaded { tex4ht }
238   {
239     \cs_set_protected:Npn \__siunitx_print_math_auxii:n #1
240     {
241       \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
242       \exp_args:NNnx \tl_replace_all:Nnn \l__siunitx_print_tmp_tl
243       { ^ } { \token_to_str:N ^ }

```

```

244         \ensuremath { \exp_args:NV \__siunitx_print_math_auxiii:n \l__siunitx_print_tmp_t
245     }
246 }
247 { }
248 }

```

(End definition for `\siunitx_print_math:n` and others. This function is documented on page 65.)

`\siunitx_print_text:n`

Typesetting in text mode is easy in font control terms but more tricky in the manipulation of the input. The easy part comes first.

```

\__siunitx_print_text_replace:n
\__siunitx_print_text_replace:N
\__siunitx_print_text_replace:NNn
\__siunitx_print_text_sub:n
\__siunitx_print_text_super:n
\__siunitx_print_text_scripts:NnN
\__siunitx_print_text_scripts:
\__siunitx_print_text_scripts_one:NnN
\__siunitx_print_text_scripts_two:NnNn
\__siunitx_print_text_scripts_two:nn
\__siunitx_print_text_scripts_two:n
249 \cs_new_protected:Npn \siunitx_print_text:n #1
250 {
251     \text
252     {
253         \bool_if:NT \l__siunitx_print_text_family_bool
254         {
255             \fontfamily { \familydefault }
256             \selectfont
257         }
258         \bool_if:NT \l__siunitx_print_text_series_bool
259         {
260             \fontseries { \seriesdefault }
261             \selectfont
262         }
263         \bool_if:NT \l__siunitx_print_text_shape_bool
264         {
265             \fontshape { \shapedefault }
266             \selectfont
267         }
268         \__siunitx_print_text_replace:n {#1}
269     }
270 }

```

To get math mode material to print in text mode, various search-and-replace steps are needed.

```

271 \cs_new_protected:Npn \__siunitx_print_text_replace:n #1
272 {
273     \group_begin:
274     \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
275     \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
276     \tl_use:N \l__siunitx_print_tmp_tl
277     \group_end:
278 }
279 \cs_new_protected:Npx \__siunitx_print_text_replace:N #1
280 {
281     \__siunitx_print_replace_font:N #1
282     \exp_not:N \__siunitx_print_text_replace:NNn #1
283     \exp_not:N \mathord { }
284     \exp_not:N \pm
285     { \exp_not:N \textpm }
286     \exp_not:N \mp
287     { \exp_not:n { \ensuremath { \mp } } } }
288 -
289     { \exp_not:N \textminus }
290     \char_generate:nn { ‘\_ } { 8 }

```

```

291     { \exp_not:N \__siunitx_print_text_sub:n }
292     ~
293     { \exp_not:N \__siunitx_print_text_super:n }
294     \exp_not:N \q_recursion_tail
295     { ? }
296     \exp_not:N \q_recursion_stop
297   }
298 \cs_new_protected:Npn \__siunitx_print_text_replace:NNn #1#2#3
299 {
300   \quark_if_recursion_tail_stop:N #2
301   \tl_replace_all:Nnn #1 {#2} {#3}
302   \__siunitx_print_text_replace:NNn #1
303 }

```

When the bidi package is loaded, we need to make sure that `\text` is doing the correct thing.

```

304 \sys_if_engine_xetex:T
305 {
306   \AtBeginDocument
307   {
308     \ifpackageloaded { bidi }
309     {
310       \cs_set_protected:Npn \__siunitx_print_text_replace:n #1
311       {
312         \group_begin:
313         \tl_set:Nn \l__siunitx_print_tmp_tl {#1}
314         \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
315         \LRE { \tl_use:N \l__siunitx_print_tmp_tl }
316         \group_end:
317       }
318     }
319   }
320 }
321 }

```

Sub- and superscripts can be in any order in the source. The first step of handling them is therefore to do a look-ahead to work out whether only one or both are present.

```

322 \cs_new_protected:Npn \__siunitx_print_text_sub:n #1
323 {
324   \__siunitx_print_text_scripts:NnN
325   \textsubscript {#1} \__siunitx_print_text_super:n
326 }
327 \cs_new_protected:Npn \__siunitx_print_text_super:n #1
328 {
329   \__siunitx_print_text_scripts:NnN
330   \textsuperscript {#1} \__siunitx_print_text_sub:n
331 }
332 \cs_new_protected:Npn \__siunitx_print_text_scripts:NnN #1#2#3
333 {
334   \cs_set_protected:Npn \__siunitx_print_text_scripts:
335   {
336     \if_meaning:w \l_peek_token #3
337     \exp_after:wN \__siunitx_print_text_scripts_two:NnNn
338     \else:
339     \exp_after:wN \__siunitx_print_text_scripts_one:Nn

```

```

340     \fi:
341     #1 {#2}
342   }
343   \peek_after:Nw \__siunitx_print_text_scripts:
344 }
345 \cs_new_protected:Npn \__siunitx_print_text_scripts: { }

```

In the simple case of one script item, we have to do a search-and-replace to deal with anything inside the argument.

```

346 \cs_new_protected:Npn \__siunitx_print_text_scripts_one:Nn #1#2
347 {
348   \group_begin:
349   \tl_set:Nn \l__siunitx_print_tmp_tl {#2}
350   \__siunitx_print_text_replace:N \l__siunitx_print_tmp_tl
351   \exp_args:NNV \group_end:
352   #1 \l__siunitx_print_tmp_tl
353 }

```

For the two scripts case, we cannot use `\textsubscript/\textsuperscript` as they don't stack directly. Instead, we sort out the ordering then use an implementation for both parts that is the same as the kernel text scripts.

```

354 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:NnNn #1#2#3#4
355 {
356   \cs_if_eq:NNTF #1 \textsubscript
357   { \__siunitx_print_text_scripts_two:nn {#4} {#2} }
358   { \__siunitx_print_text_scripts_two:nn {#2} {#4} }
359 }
360 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:nn #1#2
361 {
362   \group_begin:
363   \exp_not:N \m@th
364   \exp_not:N \ensuremath
365   {
366     ^ { \exp_not:N \__siunitx_print_text_scripts_two:n {#1} }
367     \char_generate:nn { '\_ } { 8 }
368     { \exp_not:N \__siunitx_print_text_scripts_two:n {#2} }
369   }
370   \group_end:
371 }
372 \cs_new_protected:Npn \__siunitx_print_text_scripts_two:n #1
373 {
374   \mbox
375   {
376     \fontsize \sf@size \z@ \selectfont
377     \__siunitx_print_text_scripts_one:Nn \use:n {#1}
378   }
379 }

```

(End definition for `\siunitx_print_text:n` and others. This function is documented on page 65.)

2.3 Standard settings for module options

Some of these follow naturally from the point of definition (e.g. boolean variables are always `false` to begin with), but for clarity everything is set here.


```

380 \keys_set:nn { siunitx }
381 {
382     color                =          ,
383     mode                  = math    ,
384     number-color          =          ,
385     number-mode           = math    ,
386     propagate-math-font   = false   ,
387     reset-math-version    = true     ,
388     reset-text-shape       = true     ,
389     reset-text-series     = true     ,
390     reset-text-family     = true     ,
391     text-family-to-math   = false    ,
392     text-weight-to-math   = false    ,
393     unit-color            =          ,
394     unit-mode              = math
395 }

```

These are separate as they all fall inside the same key.

```

396 \keys_set:nn { siunitx / weight-version-mapping }
397 {
398     ul = light  ,
399     el = light  ,
400     l  = light  ,
401     sl = light  ,
402     m  = normal ,
403     sb = bold   ,
404     b  = bold   ,
405     eb = bold   ,
406     ub = bold
407 }
408 </package>

```

Part VI

siunitx-quantity — Quantities

<hr/> <hr/>	<code>\siunitx_quantity:nn</code>	<code>\siunitx_quantity:nn {\langle number \rangle} {\langle unit \rangle}</code>	Parses the $\langle number \rangle$ and the $\langle unit \rangle$ as detailed for <code>\siunitx_number_parse:nN</code> and <code>\siunitx_unit_format:nN</code> , then prints the results using <code>\siunitx_print:nn</code> .
<hr/> <hr/>	<code>allow-quantity-breaks</code>	<code>allow-quantity-breaks = true false</code>	
<hr/> <hr/>	<code>prefix-mode</code>	<code>prefix-mode = power symbol</code>	
<hr/> <hr/>	<code>quantity-product</code>	<code>quantity-product = \langle tokens \rangle</code>	

1 siunitx-quantity implementation

Start the DocStrip guards.

```
1 \*package
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 \@@=siunitx_quantity
```

1.1 Initial set-up

Scratch space.

```
\l__siunitx_quantity_tmp_fp
\l__siunitx_quantity_tmp_tl
3 \tl_new:N \l__siunitx_quantity_tmp_fp
4 \tl_new:N \l__siunitx_quantity_tmp_tl
```

(End definition for `\l__siunitx_quantity_tmp_fp` and `\l__siunitx_quantity_tmp_tl`.)

1.2 Main formatting routine

```
\l__siunitx_quantity_break_bool
\l__siunitx_quantity_prefix_bool
\l__siunitx_quantity_product_tl
5 \bool_new:N \l__siunitx_quantity_prefix_bool
6 \keys_define:nn { siunitx }
7 {
8   allow-quantity-breaks .bool_set:N =
9     \l__siunitx_quantity_break_bool ,
10  prefix-mode .choice:,
11  prefix-mode / power .code:n =
12    { \bool_set_false:N \l__siunitx_quantity_prefix_bool } ,
13  prefix-mode / symbol .code:n =
14    { \bool_set_true:N \l__siunitx_quantity_prefix_bool } ,
15  quantity-product .tl_set:N =
16    \l__siunitx_quantity_product_tl
17 }
```

(End definition for `\l__siunitx_quantity_break_bool`, `\l__siunitx_quantity_prefix_bool`, and `\l__siunitx_quantity_product_tl`.)

```
\l__siunitx_quantity_number_tl
\l__siunitx_quantity_unit_tl 18 \tl_new:N \l__siunitx_quantity_number_tl
19 \tl_new:N \l__siunitx_quantity_unit_tl

(End definition for \l__siunitx_quantity_number_tl and \l__siunitx_quantity_unit_tl.)
```

`\siunitx_quantity:nn`

```
20 \cs_new_protected:Npn \siunitx_quantity:nn #1#2
21 {
22   \group_begin:
23     \siunitx_unit_options_apply:n {#2}
24     \bool_lazy_or:nnTF
25       { \l__siunitx_quantity_prefix_bool }
26       { ! \l_siunitx_number_parse_bool }
27     {
28       \siunitx_number_format:nN {#1} \l__siunitx_quantity_number_tl
29       \siunitx_unit_format:nN {#2} \l__siunitx_quantity_unit_tl
30     }
31     { \__siunitx_quantity_combine_prefix:nn {#1} {#2} }
32     \siunitx_print:nV { number } \l__siunitx_quantity_number_tl
33     \tl_use:N \l__siunitx_quantity_product_tl
34     \bool_if:NF \l__siunitx_quantity_break_bool { \nobreak }
35     \siunitx_print:nV { unit } \l__siunitx_quantity_unit_tl
36   \group_end:
37 }
```

(End definition for `\siunitx_quantity:nn`. This function is documented on page 78.)

`__siunitx_quantity_combine_prefix:nn` Combining a prefix into the main number is basically the same as the parsed branch of `\siunitx_number_format:nN`.

```
38 \cs_new_protected:Npn \__siunitx_quantity_combine_prefix:nn #1#2
39 {
40   \siunitx_unit_format:nN {#2}
41   \l__siunitx_quantity_unit_tl \l__siunitx_quantity_tmp_fp
42   \siunitx_number_parse:nN {#1} \l__siunitx_quantity_tmp_tl
43   \exp_args:NV \__siunitx_quantity_adjust_exp:nN
44     \l__siunitx_quantity_tmp_tl \l__siunitx_quantity_tmp_fp \l__siunitx_quantity_tmp_tl
45   \siunitx_number_process:NN \l__siunitx_quantity_tmp_tl \l__siunitx_quantity_tmp_tl
46   \tl_set:Nx \l__siunitx_quantity_number_tl
47     { \siunitx_number_format:N \l__siunitx_quantity_tmp_tl }
48 }
```

(End definition for `__siunitx_quantity_combine_prefix:nn`.)

`_siunitx_quantity_adjust_exp:nN` To adjust the exponent part for a combined prefix, we decompose the value, just changing the exponent part. This allows things to work with an uncertainty.

```
\_siunitx_quantity_adjust_exp:nnnnnnnnN
\_siunitx_quantity_adjust_exp:n 49 \cs_new_protected:Npn \__siunitx_quantity_adjust_exp:nN #1#2#3
50 { \__siunitx_quantity_adjust_exp:nnnnnnnnN #1 #2 #3 }
51 \cs_new_protected:Npn \__siunitx_quantity_adjust_exp:nnnnnnnnN #1#2#3#4#5#6#7#8#9
52 {
53   \tl_set:Nx #9
54   {
```

```

55      {#1} {#2} {#3} {#4} {#5}
56      \exp_args:Ne \_siunitx_quantity_adjust_exp:n { \fp_eval:n { #6#7 + #8 } }
57    }
58  }
59  \cs_new:Npn \_siunitx_quantity_adjust_exp:n #1
60    { \_siunitx_quantity_adjust_exp:Nw #1 \q_stop }
61  \cs_new:Npn \_siunitx_quantity_adjust_exp:Nw #1#2 \q_stop
62    {
63      \token_if_eq_meaning:NNTF #1 -
64        { {#1} {#2} }
65        { { } {#1#2} }
66    }

```

(End definition for `_siunitx_quantity_adjust_exp:n` and others.)

1.3 Standard settings for module options

Some of these follow naturally from the point of definition (*e.g.* boolean variables are always `false` to begin with), but for clarity everything is set here.

```

67 \keys_set:nn { siunitx }
68 {
69   allow-quantity-breaks = false ,
70   prefix-mode           = symbol ,
71   quantity-product      = \,
72 }

```

1.4 Adjustments to units

`\degree` The `\degree` unit is re-declared here: this is needed for using it in quantities. This is done here as it avoids a dependency in `siunitx-unit` on options it does not contain.

```

73 \siunitx_declare_unit:Nnn \degree { { } ^ { \circ } }
74 { quantity-product = { } }

```

(End definition for `\degree`. This function is documented on page [110](#).)

```

75 \endpackage

```

Part VII

siunitx-symbol – Symbol-related settings

1 siunitx-symbol implementation

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2 <@@=siunitx-symbol>
```

```
\l__siunitx_symbol_tmpa_tl
```

Scratch space.

```
\l__siunitx_symbol_tmpb_tl
```

```
3 \tl_new:N \l__siunitx_symbol_tmpa_tl
```

```
4 \tl_new:N \l__siunitx_symbol_tmpb_tl
```

(End definition for `\l__siunitx_symbol_tmpa_tl` and `\l__siunitx_symbol_tmpb_tl`.)

Some of the TS1 encoding is needed to provide symbols in text mode. Some of this is only for 8-bit engines, but there are places we need this even with Unicode engines. If the user has not loaded the encoding themselves, it is done here before creating the required commands.

```
5 \AtBeginDocument
```

```
6 {
```

```
7   \cs_if_free:cT { T@TS1 }
```

```
8   {
```

```
9     \DeclareFontEncoding { TS1 } { } { }
```

```
10    \DeclareFontSubstitution { TS1 } { cmr } { m } { n }
```

```
11  }
```

```
12 }
```

```
\__siunitx_symbol_textmu: For 8-bit engines, we need an upright mu.
```

```
13 \bool_lazy_or:nnF
```

```
14 { \sys_if_engine luatex_p: }
```

```
15 { \sys_if_engine xetex_p: }
```

```
16 {
```

```
17   \cs_if_free:NTF \textmu
```

```
18   {
```

```
19     \DeclareTextSymbolDefault \__siunitx_symbol_textmu: { TS1 }
```

```
20     \DeclareTextSymbol \__siunitx_symbol_textmu: { TS1 } { "00B5 }
```

```
21   }
```

```
22   { \cs_new_eq:NN \__siunitx_symbol_textmu: \textmu }
```

```
23 }
```

(End definition for `__siunitx_symbol_textmu:`.)

```
\__siunitx_symbol_mathOmega: Abstracted out to allow for safe redefinition.
```

```
24 \bool_lazy_or:nnF
```

```
25 { \sys_if_engine luatex_p: }
```

```
26 { \sys_if_engine xetex_p: }
```

```

27 {
28   \AtBeginDocument
29   {
30     \cs_new_protected:Npx \__siunitx_symbol_mathOmega:
31     {
32       \exp_not:N \ensuremath
33       {
34         \cs_if_exist:NTF \upOmega
35         { \exp_not:N \upOmega }
36         { \exp_not:N \Omega }
37       }
38     }
39   }
40 }

```

(End definition for `__siunitx_symbol_mathOmega:`.)

`__siunitx_symbol_non_latin:n` As in `siunitx-unit`, but internal in both cases as it's rather specialised.
`__siunitx_symbol_non_latin:nnnn`

```

41 \bool_lazy_or:nnTF
42 { \sys_if_engine luatex_p: }
43 { \sys_if_engine xetex_p: }
44 {
45   \cs_new:Npn \__siunitx_symbol_non_latin:n #1
46   { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
47 }
48 {
49   \cs_new:Npn \__siunitx_symbol_non_latin:n #1
50   {
51     \exp_last_unbraced:Nf \__siunitx_symbol_non_latin:nnnn
52     { \char_codepoint_to_bytes:n {#1} }
53   }
54   \cs_new:Npn \__siunitx_symbol_non_latin:nnnn #1#2#3#4
55   {
56     \exp_after:wN \exp_after:wN \exp_after:wN
57     \exp_not:N \char_generate:nn {#1} { 13 }
58     \exp_after:wN \exp_after:wN \exp_after:wN
59     \exp_not:N \char_generate:nn {#2} { 13 }
60   }
61 }

```

(End definition for `__siunitx_symbol_non_latin:n` and `__siunitx_symbol_non_latin:nnnn`.)

`__siunitx_symbol_if_replace:NnT` A test to see if the unit definition which applies is still one we expect: here that means it is just using a (Unicode) codepoint. The comparison is string-based as `unicode-math` (at least) can alter some of them.

```

62 \prg_new_protected_conditional:Npnn \__siunitx_symbol_if_replace:Nn #1#2 { T , TF }
63 {
64   \group_begin:
65   \tl_set:Nx \l__siunitx_symbol_tmpa_tl { \__siunitx_symbol_non_latin:n {#2} }
66   \protected@edef \l__siunitx_symbol_tmpa_tl
67   { \exp_not:N \mathrm { \l__siunitx_symbol_tmpa_tl } }
68   \keys_set:nn { siunitx } { parse-units = false }
69   \siunitx_unit_format:nN {#1} \l__siunitx_symbol_tmpb_tl
70   \str_if_eq:VVTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl

```

```

71     {
72       \group_end:
73       \prg_return_true:
74     }
75     {
76       \group_end:
77       \prg_return_false:
78     }
79   }

```

(End definition for `_siunitx_symbol_if_replace:NnT`.)

At the start of the document, fonts are fixed and the user may have altered unit set up. If things are unchanged, we can alter the settings such that they use something “more sensible”.

```

80 \AtBeginDocument
81   {

```

For `\angstrom`, direct input works in text mode so there is only a need to tidy up for math mode. If `fontspec` is loaded then that problem goes away, so nothing needs to be done.

```

82   \_siunitx_symbol_if_replace:NnT \angstrom { "00C5 }
83   {
84     \@ifpackageloaded { fontspec }
85     { }
86     {
87       \siunitx_declare_unit:Nx \angstrom
88       {
89         \siunitx_print_text:n
90         { \_siunitx_symbol_non_latin:n { "00C5 } }
91       }
92     }
93   }
94   \_siunitx_symbol_if_replace:NnT \ohm { "03A9 }
95   {
96     \siunitx_declare_unit:Nx \ohm
97     {
98       \bool_lazy_or:nnTF
99       { \sys_if_engine luatex_p: }
100      { \sys_if_engine xetex_p: }
101      {
102        \siunitx_print_text:n
103        { \_siunitx_symbol_non_latin:n { "03A9 } }
104      }
105      { \_siunitx_symbol_mathOmega: }
106    }
107  }
108  \_siunitx_symbol_if_replace:NnT \micro { "03BC }
109  {
110    \siunitx_declare_prefix:Nnx \micro { -6 }
111    {
112      \siunitx_print_text:n
113      {
114        \bool_lazy_or:nnTF
115        { \sys_if_engine luatex_p: }

```

```

116         { \sys_if_engine_xetex_p: }
117         { \__siunitx_symbol_non_latin:n { "00B5 } }
118         { \exp_not:N \__siunitx_symbol_textmu: }
119     }
120 }
121 }
122 }

```

`__siunitx_symbol_texttimes:` For the times symbol, only LuaTeX allows us to use the math mode symbol directly. However, that likely won't follow the surrounding font appearance, so in all cases we load the TS1 version for text. Otherwise much the same as `\textmu` support. It's hard to check for the product symbol, so we just go with it an hope for the best!

```

123 \AtBeginDocument
124 {
125     \cs_if_free:NTF \texttimes
126     {
127         \DeclareTextSymbolDefault \__siunitx_symbol_texttimes: { TS1 }
128         \DeclareTextSymbol \__siunitx_symbol_texttimes: { TS1 } { "00D6 }
129     }
130     { \cs_new_eq:NN \__siunitx_symbol_texttimes: \texttimes }
131     \group_begin:
132         \tl_set:Nn \l__siunitx_symbol_tmpa_tl
133             { { } { } { 2 } { } { } { } { 1 } }
134         \tl_set:Nx \l__siunitx_symbol_tmpa_tl
135             { \siunitx_number_format:N \l__siunitx_symbol_tmpa_tl }
136         \tl_set:Nn \l__siunitx_symbol_tmpb_tl { 2 \times 10 ^ { 1 } }
137         \tl_if_eq:NNTF \l__siunitx_symbol_tmpa_tl \l__siunitx_symbol_tmpb_tl
138         {
139             \group_end:
140             \keys_set:nn { siunitx }
141             {
142                 exponent-product =
143                 \ifmmode \times \else \__siunitx_symbol_texttimes: \fi ,
144                 product-symbol =
145                 \ifmmode \times \else \__siunitx_symbol_texttimes: \fi
146             }
147         }
148         { \group_end: }
149     }

```

(End definition for __siunitx_symbol_texttimes:.)

1.1 Bookmark definitions

Inside PDF strings, the definitions for symbols need to be reset: we keep things to a minimum and just reset the functions we know are used above.

```

150 \AtBeginDocument
151 {
152     \@ifpackageloaded { hyperref }
153     {
154         \pdfstringdefDisableCommands
155         {
156             \cs_set_eq:NN \siunitx_print_text:n \use:n

```



```

157         \cs_set:Npx \__siunitx_symbol_textmu:
158             { \__siunitx_symbol_non_latin:n { "00B5 } }
159         \cs_set:Npx \__siunitx_symbol_mathOmega:
160             { \__siunitx_symbol_non_latin:n { "03A9 } }
161         \siunitx_declare_unit:Nx \degree
162             { \__siunitx_symbol_non_latin:n { "00B0 } }
163         \siunitx_declare_unit:Nx \degreeCelsius
164             { \__siunitx_symbol_non_latin:n { "00B0 } C }
165     }
166 }
167 { }
168 }
169 </package>

```

Part VIII

siunitx-table – Formatting numbers in tables

1 Numbers in tables

`\siunitx_cell_begin:w`
`\siunitx_cell_end:`

`\siunitx_cell_begin:w` $\langle preamble \rangle$ `\ignorespaces`
 $\langle content \rangle$
`\siunitx_cell_end:`

Collects the $\langle preamble \rangle$ and $\langle content \rangle$ tokens, and determines if it is text or a number (as parsed by `\siunitx_number_parse:nN`). It produces output of a fixed width suitable for alignment in a table, although it is not *required* that the code is used within a cell. Note that `\ignorespaces` must occur in the “cell”: it marks the end of the T_EX `\halign` template.

1.1 Key–value options

The options defined by this submodule are available within the l3keys `siunitx` tree.

`table-align-comparator`

`table-align-comparator = true|false`

`table-align-exponent`

`table-align-exponent = true|false`

`table-align-text-after`

`table-align-text-after = true|false`

`table-align-text-before`

`table-align-text-before = true|false`

`table-align-uncertainty`

`table-align-uncertainty = true|false`

`table-alignment`

`table-alignment = center|left|right`

`table-alignment-mode`

`table-alignment-mode = format|marker|none`

`table-auto-round`

`table-auto-round = true|false`

`table-column-width`

`table-column-width = $\langle width \rangle$`

`table-fixed-width`

`table-fixed-width = true|false`

<hr/> <hr/> table-format	table-format = $\langle format \rangle$
<hr/> <hr/> table-number-alignment	table-number-alignment = center left right
<hr/> <hr/> table-text-alignment	table-text-alignment = center left right

2 siunitx-table implementation

Start the DocStrip guards.

```
1  $\langle *package \rangle$ 
```

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2  $\langle @@=siunitx\_table \rangle$ 
```

```
 $\backslash l\_siunitx\_table\_tmp\_box$   
 $\backslash l\_siunitx\_table\_tmp\_dim$   
 $\backslash l\_siunitx\_table\_tmp\_tl$ 
```

Scratch space.

```
3  $\backslash box\_new:N \backslash l\_siunitx\_table\_tmp\_box$   
4  $\backslash dim\_new:N \backslash l\_siunitx\_table\_tmp\_dim$   
5  $\backslash tl\_new:N \backslash l\_siunitx\_table\_tmp\_tl$ 
```

(End definition for $\backslash l_siunitx_table_tmp_box$, $\backslash l_siunitx_table_tmp_dim$, and $\backslash l_siunitx_table_tmp_tl$.)

2.1 Interface functions

```
 $\backslash l\_siunitx\_table\_text\_bool$ 
```

Used to track that a cell is purely text.

```
6  $\backslash bool\_new:N \backslash l\_siunitx\_table\_text\_bool$ 
```

(End definition for $\backslash l_siunitx_table_text_bool$.)

```
 $\backslash siunitx\_cell\_begin:w$   
 $\backslash siunitx\_cell\_end:$ 
```

The start and end of the cell need to deal with the possibility of a cell containing only text.

```
7  $\backslash cs\_new\_protected:Npn \backslash siunitx\_cell\_begin:w$   
8 {  
9    $\backslash bool\_set\_false:N \backslash l\_siunitx\_table\_text\_bool$   
10   $\backslash bool\_if:NTF \backslash l\_siunitx\_number\_parse\_bool$   
11    {  $\backslash \_siunitx\_table\_collect\_begin:$  }  
12    {  $\backslash \_siunitx\_table\_direct\_begin:$  }  
13 }  
14  $\backslash cs\_new\_protected:Npn \backslash siunitx\_cell\_end:$   
15 {  
16    $\backslash bool\_if:NF \backslash l\_siunitx\_table\_text\_bool$   
17   {  
18      $\backslash bool\_if:NTF \backslash l\_siunitx\_number\_parse\_bool$   
19     {  $\backslash \_siunitx\_table\_collect\_end:$  }  
20     {  $\backslash \_siunitx\_table\_direct\_end:$  }  
21   }  
22 }
```

(End definition for $\backslash siunitx_cell_begin:w$ and $\backslash siunitx_cell_end:$. These functions are documented on page 86.)

2.2 Collecting tokens

<code>\l__siunitx_table_collect_tl</code>	Space for tokens.
23	<code>\tl_new:N \l__siunitx_table_collect_tl</code>
	(End definition for <code>\l__siunitx_table_collect_tl</code> .)
<code>__siunitx_table_collect_begin:</code>	Collecting a tabular cell means doing a token-by-token collection. In previous versions of <code>siunitx</code> that was done along with picking out the numerical part, but the code flow ends up very tricky. Here, therefore, we just collect up the unchanged tokens first. The definition of <code>\cr</code> is used to allow collection of any tokens inserted after the main content when dealing with the last cell of a row: the “group” around it is needed to avoid issues with the underlying <code>\halign</code> . (The approach is based on that in <code>colcell</code> .) Notice that as each cell forms a group there is no need to reset the definition of <code>\cr</code> . We use an auxiliary to fish out the <code>\ignorespaces</code> from the template: that has to go to avoid issues with the peek-ahead code (everything before the <code>#</code> needs to be read <i>before</i> the Appendix D trick gets applied). Some packages add additional tokens before the <code>\ignorespaces</code> , which are dealt with by the delimited argument.
<code>__siunitx_table_collect_begin:w</code>	
24	<code>\cs_new_protected:Npn __siunitx_table_collect_begin:</code>
25	<code>{</code>
26	<code>\tl_clear:N \l__siunitx_table_collect_tl</code>
27	<code>\if_false: { \fi:</code>
28	<code>\cs_set_protected:Npn \cr</code>
29	<code>{</code>
30	<code>__siunitx_table_collect_loop:</code>
31	<code>\tex_cr:D</code>
32	<code>}</code>
33	<code>\if_false: } \fi:</code>
34	<code>__siunitx_table_collect_begin:w</code>
35	<code>}</code>
36	<code>\cs_new_protected:Npn __siunitx_table_collect_begin:w #1 \ignorespaces</code>
37	<code>{ __siunitx_table_collect_loop: #1 }</code>
	(End definition for <code>__siunitx_table_collect_begin:</code> and <code>__siunitx_table_collect_begin:w</code> .)
<code>__siunitx_table_collect_loop:</code>	Collecting up the cell content needs a loop: this is done using a peek approach as it’s most natural. (A slower approach is possible using something like the <code>\tl_lower_case:n</code> loop code.) The set of possible tokens is somewhat limited compared to an arbitrary cell
<code>__siunitx_table_collect_group:n</code>	(<i>cf.</i> the approach in <code>colcell</code>): the special cases are pulled out for manual handling. The flexible lookup approach is more-or-less the same idea as in the kernel <code>case</code> functions.
<code>__siunitx_table_collect_token:N</code>	The <code>\relax</code> special case covers the case where <code>\\</code> has been expanded in an empty cell.
<code>__siunitx_table_collect_search:NnF</code>	
<code>__siunitx_table_collect_search_aux:NNn</code>	
38	<code>\cs_new_protected:Npn __siunitx_table_collect_loop:</code>
39	<code>{</code>
40	<code>\peek_catcode_ignore_spaces:NTF \c_group_begin_token</code>
41	<code>{ __siunitx_table_collect_group:n }</code>
42	<code>{ __siunitx_table_collect_token:N }</code>
43	<code>}</code>
44	<code>\cs_new_protected:Npn __siunitx_table_collect_group:n #1</code>
45	<code>{</code>
46	<code>\tl_put_right:Nn \l__siunitx_table_collect_tl { {#1} }</code>
47	<code>__siunitx_table_collect_loop:</code>
48	<code>}</code>
49	<code>\cs_new_protected:Npn __siunitx_table_collect_token:N #1</code>

```

50 {
51   \__siunitx_table_collect_search:NnF #1
52   {
53     \unskip          { \__siunitx_table_collect_loop: }
54     \end              { \tabularnewline \end }
55     \relax           { \relax }
56     \tabularnewline  { \tabularnewline }
57     \siunitx_cell_end: { \siunitx_cell_end: }
58   }
59   {
60     \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
61     \__siunitx_table_collect_loop:
62   }
63 }
64 \AtBeginDocument
65 {
66   \ifpackageloaded { mdwtab }
67   {
68     \cs_set_protected:Npn \__siunitx_table_collect_token:N #1
69     {
70       \__siunitx_table_collect_search:NnF #1
71       {
72         \@maybe@unskip { \__siunitx_table_collect_loop: }
73         \tab@setcr      { \__siunitx_table_collect_loop: }
74         \unskip         { \__siunitx_table_collect_loop: }
75         \end            { \tabularnewline \end }
76         \relax          { \relax }
77         \tabularnewline { \tabularnewline }
78         \siunitx_cell_end: { \siunitx_cell_end: }
79       }
80       {
81         \tl_put_right:Nn \l__siunitx_table_collect_tl {#1}
82         \__siunitx_table_collect_loop:
83       }
84     }
85   }
86   { }
87 }
88 \cs_new_protected:Npn \__siunitx_table_collect_search:NnF #1#2#3
89 {
90   \__siunitx_table_collect_search_aux:NNn #1
91   #2
92   #1 {#3}
93   \q_stop
94 }
95 \cs_new_protected:Npn \__siunitx_table_collect_search_aux:NNn #1#2#3
96 {
97   \token_if_eq_meaning:NNTF #1 #2
98   { \use_i_delimit_by_q_stop:nw {#3} }
99   { \__siunitx_table_collect_search_aux:NNn #1 }
100 }

```

(End definition for __siunitx_table_collect_loop: and others.)

2.3 Separating collected material

The input needs to be divided into numerical tokens and those which appear before and after them. This needs a second loop and validation.

```

\l__siunitx_table_before_tl Space for tokens.
\l__siunitx_table_number_tl 101 \tl_new:N \l__siunitx_table_before_tl
\l__siunitx_table_after_tl 102 \tl_new:N \l__siunitx_table_number_tl
                             103 \tl_new:N \l__siunitx_table_after_tl

(End definition for \l__siunitx_table_before_tl, \l__siunitx_table_number_tl, and \l__siunitx_
table_after_tl.)

\__siunitx_table_collect_end: At the end of the cell, expand all of the content as far as possible then split it up into
numerical and non-numerical parts.
104 \cs_new_protected:Npn \__siunitx_table_collect_end:
105 {
106   \protected@edef \l__siunitx_table_collect_tl
107   { \l__siunitx_table_collect_tl }
108   \exp_args:NV \__siunitx_table_split:nNNN
109   \l__siunitx_table_collect_tl
110   \l__siunitx_table_before_tl
111   \l__siunitx_table_number_tl
112   \l__siunitx_table_after_tl
113   \tl_if_empty:NTF \l__siunitx_table_number_tl
114   { \__siunitx_table_print_text:V \l__siunitx_table_before_tl }
115   {
116     \__siunitx_table_print:VVV
117     \l__siunitx_table_before_tl
118     \l__siunitx_table_number_tl
119     \l__siunitx_table_after_tl
120   }
121 }

(End definition for \__siunitx_table_collect_end:.)

\__siunitx_table_split:nNNN Splitting into parts uses the fact that numbers cannot contain groups and that we can
\__siunitx_table_split_loop:NNN track where we are up to based on the content of the token lists.
\__siunitx_table_split_group:NNNn
\__siunitx_table_split_token:NNNN
122 \cs_new_protected:Npn \__siunitx_table_split:nNNN #1#2#3#4
123 {
124   \tl_clear:N #2
125   \tl_clear:N #3
126   \tl_clear:N #4
127   \__siunitx_table_split_loop:NNN #2#3#4 #1 \q_recursion_tail \q_recursion_stop
128   \__siunitx_table_split_tidy:N #2
129   \__siunitx_table_split_tidy:N #4
130 }
131 \cs_new_protected:Npn \__siunitx_table_split_loop:NNN #1#2#3
132 {
133   \peek_catcode_ignore_spaces:NTF \c_group_begin_token
134   { \__siunitx_table_split_group:NNNn #1#2#3 }
135   { \__siunitx_table_split_token:NNNN #1#2#3 }
136 }
137 \cs_new_protected:Npn \__siunitx_table_split_group:NNNn #1#2#3#4
138 {

```

```

139 \tl_if_empty:NTF #2
140 { \tl_put_right:Nn #1 { {#4} } }
141 { \tl_put_right:Nn #3 { {#4} } }
142 \__siunitx_table_split_loop:NNN #1#2#3
143 }
144 \cs_new_protected:Npn \__siunitx_table_split_token:NNNN #1#2#3#4
145 {
146 \quark_if_recursion_tail_stop:N #4
147 \tl_if_empty:NTF \l__siunitx_table_after_tl
148 {
149 \siunitx_if_number_token:NTF #4
150 { \tl_put_right:Nn #2 {#4} }
151 {
152 \tl_if_empty:NTF #2
153 { \tl_put_right:Nn #1 {#4} }
154 { \tl_put_right:Nn #3 {#4} }
155 }
156 }
157 { \tl_put_right:Nn #3 {#4} }
158 \__siunitx_table_split_loop:NNN #1#2#3
159 }

```

(End definition for `__siunitx_table_split:nNNN` and others.)

```

\__siunitx_table_split_tidy:N
\__siunitx_table_split_tidy:Nn
\__siunitx_table_split_tidy:NV

```

A quick test for the entire content being surrounded by a set of braces: rather than look explicitly, use the fact that a string comparison can detect the same thing. The auxiliary is needed to avoid having to go *via* a `:D` function (for the expansion behaviour).

```

160 \cs_new_protected:Npn \__siunitx_table_split_tidy:N #1
161 {
162 \tl_if_empty:NF #1
163 { \__siunitx_table_split_tidy:NV #1 #1 }
164 }
165 \cs_new_protected:Npn \__siunitx_table_split_tidy:Nn #1#2
166 {
167 \str_if_eq:onT { \exp_after:wN { \use:n #2 } } {#2}
168 { \tl_set:Nn #1 { \use:n #2 } }
169 }
170 \cs_generate_variant:Nn \__siunitx_table_split_tidy:Nn { NV }

```

(End definition for `__siunitx_table_split_tidy:N` and `__siunitx_table_split_tidy:Nn`.)

2.4 Printing numbers in cells: spacing

Getting the general alignment correct in tables is made more complex than one would like by the `colortbl` package. In the original L^AT_EX 2_ε definition, cell material is centred by a construction of the (primitive) form

```

\hfil
#
\hfil

```

which only uses `fil` stretch. That is altered by `colortbl` to broadly

```

\hskip 0pt plus 0.5fill
\kern 0pt
#
\hskip 0pt plus 0.5fill

```

which means there is fill stretch to worry about and the kern as well.

`_siunitx_table_skip:n` To prevent combination of skips, a kern is inserted after each one. This is best handled as a short auxiliary.

```

171 \cs_new_protected:Npn \_siunitx\_table\_skip:n #1
172 {
173   \skip_horizontal:n {#1}
174   \tex_kern:D \c_zero_skip
175 }

```

(End definition for `_siunitx_table_skip:n`.)

`\l_siunitx_table_column_width_dim` Settings which apply to aligned columns in general.

```

\l\_siunitx\_table\_fixed\_width\_bool 176 \keys_define:nn { siunitx }
177 {
178   table-column-width .dim_set:N =
179     \l\_siunitx\_table\_column\_width\_dim ,
180   table-fixed-width .bool_set:N =
181     \l\_siunitx\_table\_fixed\_width\_bool
182 }

```

(End definition for `\l_siunitx_table_column_width_dim` and `\l_siunitx_table_fixed_width_bool`.)

`_siunitx_table_align_center:n` The beginning and end of each table cell have to adjust the position of the content using glue. When `colortbl` is loaded the glue is done in two parts: one for our positioning and one to explicitly override that from the package. Using a two-step auxiliary chain avoids needing to repeat any code and the impact of the extra expansion should be trivial.

```

\_siunitx\_table\_align\_left:n
\_siunitx\_table\_align\_right:n
\_siunitx\_table\_align\_auxi:nn
\_siunitx\_table\_align\_auxii:nn
183 \cs_new_protected:Npn \_siunitx\_table\_align\_center:n #1
184 { \_siunitx\_table\_align\_auxi:nn {#1} { 0pt+plus-0.5fill } }
185 \cs_new_protected:Npn \_siunitx\_table\_align\_left:n #1
186 { \_siunitx\_table\_align\_auxi:nn {#1} { 0pt } }
187 \cs_new_protected:Npn \_siunitx\_table\_align\_right:n #1
188 { \_siunitx\_table\_align\_auxi:nn {#1} { 0pt+plus-1fill } }
189 \cs_new_protected:Npn \_siunitx\_table\_align\_auxi:nn #1#2
190 {
191   \bool_if:NTF \l\_siunitx\_table\_fixed\_width\_bool
192     { \hbox_to_wd:nn \l\_siunitx\_table\_column\_width\_dim }
193     { \use:n }
194     {
195       \_siunitx\_table\_skip:n {#2}
196       #1
197       \_siunitx\_table\_skip:n { 0pt+plus-1fill - #2 }
198     }
199 }
200 \AtBeginDocument
201 {
202   \@ifpackageloaded { colortbl }
203   {
204     \cs_new_eq:NN

```



```

205         \_siunitx_table_align_auxii:nn
206         \_siunitx_table_align_auxi:nn
207     \cs_set_protected:Npn \_siunitx_table_align_auxi:nn #1#2
208     {
209         \_siunitx_table_skip:n{ 0pt~plus~-0.5fill }
210         \_siunitx_table_align_auxii:nn {#1} {#2}
211         \_siunitx_table_skip:n { 0pt~plus~-0.5fill }
212     }
213 }
214 { }
215 }

```

(End definition for `_siunitx_table_align_center:n` and others.)

2.5 Printing just text

In cases where there is no numerical part, siunitx allows alignment of the “escaped” text independent of the underlying column type.

`\l_siunitx_table_align_text_tl` Alignment is handled using a `tl` as this allows a fast lookup at the point of use.

```

216 \keys_define:nn { siunitx }
217 {
218     table-text-alignment .choices:nn =
219     { center , left , right }
220     { \tl_set:Nn \l_siunitx_table_align_text_tl {#1} } ,
221 }
222 \tl_new:N \l_siunitx_table_align_text_tl

```

(End definition for `\l_siunitx_table_align_text_tl`.)

`_siunitx_table_print_text:n` Printing escaped text is easy: just place it in correctly in the column.

```

\_siunitx_table_print_text:V
223 \cs_new_protected:Npn \_siunitx_table_print_text:n #1
224 {
225     \bool_set_true:N \l_siunitx_table_text_bool
226     \use:c { \_siunitx_table_align_ \l_siunitx_table_align_text_tl :n } {#1}
227 }
228 \cs_generate_variant:Nn \_siunitx_table_print_text:n { V }

```

(End definition for `_siunitx_table_print_text:n`.)

2.6 Number alignment: core ideas

`\l_siunitx_table_integer_box` Boxes for the content before and after the decimal marker.

```

\_siunitx_table_decimal_box
229 \box_new:N \l_siunitx_table_integer_box
230 \box_new:N \l_siunitx_table_decimal_box

```

(End definition for `\l_siunitx_table_integer_box` and `\l_siunitx_table_decimal_box`.)

`_siunitx_table_fil:` A primitive renamed.

```

231 \cs_new_eq:NN \_siunitx_table_fil: \tex_hfil:D

```

(End definition for `_siunitx_table_fil:`.)

`_siunitx_table_cleanup_decimal:w` To remove the excess marker tokens in a decimal part.

```

232 \cs_new:Npn \_siunitx_table_cleanup_decimal:w
233   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_nil #5 \q_nil #6 \q_nil #7 \q_nil
234   { #1#2#3#4#5#6#7 }

```

(End definition for `_siunitx_table_cleanup_decimal:w`.)

`_siunitx_table_center_marker:` When centering on the decimal marker, the easiest approach is to simply re-box the two parts. That is needed whether or not we are parsing numbers, so is best as a short auxiliary.

```

235 \cs_new_protected:Npn \_siunitx_table_center_marker:
236   {
237     \dim_compare:nNnTF
238       { \box_wd:N \l__siunitx_table_integer_box }
239       > { \box_wd:N \l__siunitx_table_decimal_box }
240       {
241         \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
242         { \box_wd:N \l__siunitx_table_integer_box }
243         {
244           \hbox_unpack:N \l__siunitx_table_decimal_box
245           \_siunitx_table_fil:
246         }
247       }
248       {
249         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
250         { \box_wd:N \l__siunitx_table_decimal_box }
251         {
252           \_siunitx_table_fil:
253           \hbox_unpack:N \l__siunitx_table_integer_box
254         }
255       }
256   }

```

(End definition for `_siunitx_table_center_marker:`.)

`\l__siunitx_table_auto_round_bool` Options for tables with defined space.

`\l__siunitx_table_align_mode_tl`

`\l__siunitx_table_align_number_tl`

```

257 \keys_define:nn { siunitx }
258   {
259     table-alignment .meta:n =
260       { table-number-alignment = #1 , table-text-alignment = #1 },
261     table-alignment-mode .choices:nn =
262       { none , format , marker }
263       { \tl_set_eq:NN \l__siunitx_table_align_mode_tl \l_keys_choice_tl } ,
264     table-auto-round .bool_set:N =
265       \l__siunitx_table_auto_round_bool ,
266     table-format .code:n =
267       {
268         \_siunitx_table_split:nNNN {#1}
269         \l__siunitx_table_before_model_tl
270         \l__siunitx_table_model_tl
271         \l__siunitx_table_after_model_tl
272         \exp_args:NV \_siunitx_table_generate_model:n \l__siunitx_table_model_tl
273         \tl_set:Nn \l__siunitx_table_align_mode_tl { format }
274       } ,

```

```

275     table-number-alignment .choices:nn =
276     { center , left , right }
277     { \tl_set_eq:NN \l__siunitx_table_align_number_tl \l_keys_choice_tl }
278   }
279   \tl_new:N \l__siunitx_table_align_mode_tl
280   \tl_new:N \l__siunitx_table_align_number_tl

(End definition for \l__siunitx_table_auto_round_bool, \l__siunitx_table_align_mode_tl, and
\l__siunitx_table_align_number_tl.)

```

\l__siunitx_table_format_tl The input and output versions of the model entry in a table.

```

\l__siunitx_table_model_tl
281 \tl_new:N \l__siunitx_table_format_tl
282 \tl_new:N \l__siunitx_table_before_model_tl
283 \tl_new:N \l__siunitx_table_model_tl
284 \tl_new:N \l__siunitx_table_after_model_tl

(End definition for \l__siunitx_table_format_tl and \l__siunitx_table_model_tl.)

```

_siunitx_table_generate_model:n Creating a model for a table at this stage means parsing the format and converting that to an appropriate model. Things are quite straight-forward other than the uncertainty part. At this stage there is no point in formatting the model: that has to happen at point-of-use.

```

285 \cs_new_protected:Npn \_siunitx_table_generate_model:n #1
286 {
287   \group_begin:
288     \bool_set_true:N \l_siunitx_number_parse_bool
289     \keys_set:nn { siunitx } { track-explicit-plus = true }
290     \siunitx_number_parse:nN {#1} \l__siunitx_table_format_tl
291   \exp_args:NNNV \group_end:
292   \tl_set:Nn \l__siunitx_table_format_tl \l__siunitx_table_format_tl
293   \tl_if_empty:NF \l__siunitx_table_format_tl
294   {
295     \exp_after:wN \_siunitx_table_generate_model:nnnnnnn
296     \l__siunitx_table_format_tl
297   }
298 }
299 \cs_new_protected:Npn \_siunitx_table_generate_model:nnnnnnn #1#2#3#4#5#6#7
300 {
301   \tl_set:Nx \l__siunitx_table_model_tl
302   {
303     \exp_not:n { {#1} {#2} }
304     { \prg_replicate:nn {#3} { 8 } }
305     { \prg_replicate:nn { 0 #4 } { 8 } }
306     {
307       \tl_if_blank:NF {#5}
308       {
309         \use:c { __siunitx_table_generate_model_ \tl_head:n {#5} :nw }
310         #5
311       }
312     }
313     \exp_not:n { {#6} }
314     {
315       \int_compare:nNnTF {#7} = 0
316       { 0 }

```

```

317         { \prg_replicate:nn {#7} { 8 } }
318     }
319 }
320 }
321 \cs_new:Npn \__siunitx_table_generate_model_S:nw #1#2
322 { { S } { \prg_replicate:nn {#2} { 8 } } }

```

(End definition for `__siunitx_table_generate_model:n`, `__siunitx_table_generate_model:nnnnnnn`, and `__siunitx_table_generate_model_S:nw`.)

2.7 Directly printing without collection

Collecting the number allows for various effects but is not as fast as simply aligning on the first token that is a decimal marker. The strategy here is that used by `dcolumn`.

After removing the `\ignorespaces` at the start of the cell (see comments for `__siunitx_table_collect_begin:N`), check to see if there is a `{` and branch as appropriate.

```

\__siunitx_table_direct_begin:
\__siunitx_table_direct_begin:w
\__siunitx_table_direct_end:
\__siunitx_table_direct_marker:
\__siunitx_table_direct_marker_switch:
\__siunitx_table_direct_marker_end:
\__siunitx_table_direct_format:
\__siunitx_table_direct_format:nnnnnnn
\__siunitx_table_direct_format:w
\__siunitx_table_direct_format_switch:
\__siunitx_table_direct_format_end:
\__siunitx_table_direct_none:
\__siunitx_table_direct_none_end:
323 \cs_new_protected:Npn \__siunitx_table_direct_begin:
324 { \__siunitx_table_direct_begin:w }
325 \cs_new_protected:Npn \__siunitx_table_direct_begin:w #1 \ignorespaces
326 {
327     #1
328     \peek_catcode_ignore_spaces:NTF \c_group_begin_token
329     { \__siunitx_table_print_text:n }
330     {
331         \m@th
332         \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl : }
333     }
334 }
335 \cs_new_protected:Npn \__siunitx_table_direct_end:
336 { \use:c { __siunitx_table_direct_ \l__siunitx_table_align_mode_tl _end: } }

```

When centring the content about a decimal marker, the trick is to collect everything into two boxes and then compare the sizes. As we are always in math mode, we can use a math active token to make the switch. The up-front setting of the decimal box deals with the case where there is no decimal part.

```

337 \cs_new_protected:Npn \__siunitx_table_direct_marker:
338 {
339     \hbox_set:Nn \l__siunitx_table_tmp_box
340     { \ensuremath { \mathord { \l__siunitx_number_output_decimal_tl } } }
341     \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
342     { \box_wd:N \l__siunitx_table_tmp_box }
343     { \__siunitx_table_fil: }
344     \hbox_set:Nw \l__siunitx_table_integer_box
345     \c_math_toggle_token
346     \tl_map_inline:Nn \l__siunitx_number_input_decimal_tl
347     {
348         \char_set_active_eq:NN ##1 \__siunitx_table_direct_marker_switch:
349         \char_set_mathcode:nn { '##1 } { "8000 }
350     }
351 }
352 \cs_new_protected:Npn \__siunitx_table_direct_marker_switch:
353 {

```

```

354     \c_math_toggle_token
355   \hbox_set_end:
356   \hbox_set:Nw \l__siunitx_table_decimal_box
357     \c_math_toggle_token
358     \l_siunitx_number_output_decimal_tl
359 }
360 \cs_new_protected:Npn \__siunitx_table_direct_marker_end:
361 {
362   \c_math_toggle_token
363   \hbox_set_end:
364   \__siunitx_table_center_marker:
365   \box_use_drop:N \l__siunitx_table_integer_box
366   \box_use_drop:N \l__siunitx_table_decimal_box
367 }

```

For the version where there is space reserved, first format and decompose that, then create appropriately-sized boxes.

```

368 \cs_new_protected:Npn \__siunitx_table_direct_format:
369 {
370   \tl_set:Nx \l__siunitx_table_tmp_tl
371   { \siunitx_number_format:NN \l__siunitx_table_model_tl \q_nil }
372   \exp_after:wN \__siunitx_table_direct_format_aux:w
373   \l__siunitx_table_tmp_tl \q_stop
374 }
375 \cs_new_protected:Npn \__siunitx_table_direct_format_aux:w
376 #1 \q_nil #2 \q_nil #3 \q_stop
377 {
378   \hbox_set:Nn \l__siunitx_table_tmp_box
379   { \ensuremath { \__siunitx_table_cleanup_decimal:w #3 } }
380   \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
381   { \box_wd:N \l__siunitx_table_tmp_box }
382   { \__siunitx_table_fil: }
383   \hbox_set:Nn \l__siunitx_table_tmp_box { \ensuremath { #1#2 } }
384   \hbox_set_to_wd:Nnw \l__siunitx_table_integer_box
385   { \box_wd:N \l__siunitx_table_tmp_box }
386   \c_math_toggle_token
387   \tl_map_inline:Nn \l_siunitx_number_input_decimal_tl
388   {
389     \char_set_active_eq:NN ##1 \__siunitx_table_direct_format_switch:
390     \char_set_mathcode:nn { '##1 } { "8000 }
391   }
392   \__siunitx_table_fil:
393 }
394 \cs_new_protected:Npn \__siunitx_table_direct_format_switch:
395 {
396   \c_math_toggle_token
397   \hbox_set_end:
398   \hbox_set_to_wd:Nnw \l__siunitx_table_decimal_box
399   { \box_wd:N \l__siunitx_table_decimal_box }
400   \c_math_toggle_token
401   \mathord { \l_siunitx_number_output_decimal_tl }
402 }
403 \cs_new_protected:Npn \__siunitx_table_direct_format_end:
404 {

```

```

405     \c_math_toggle_token
406     \__siunitx_table_fil:
407 \hbox_set_end:
408 \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
409 {
410     \box_use_drop:N \l__siunitx_table_integer_box
411     \box_use_drop:N \l__siunitx_table_decimal_box
412 }
413 }

```

No parsing and no alignment is easy.

```

414 \cs_new_protected:Npn \__siunitx_table_direct_none: { \c_math_toggle_token }
415 \cs_new_protected:Npn \__siunitx_table_direct_none_end: { \c_math_toggle_token }

```

(End definition for __siunitx_table_direct_begin: and others.)

2.8 Printing numbers in cells: main functions

For alignment of text outside of a number.

```

416 \box_new:N \l__siunitx_table_before_box
417 \box_new:N \l__siunitx_table_after_box

```

(End definition for \l__siunitx_table_before_box and \l__siunitx_table_after_box.)

Used to “carry forward” the amount of white space which needs to be inserted after the decimal marker.

```

418 \dim_new:N \l__siunitx_table_carry_dim

```

(End definition for \l__siunitx_table_carry_dim.)

Alignment is handled using a tl as this allows a fast lookup at the point of use.

```

419 \keys_define:nn { siunitx }
420 {
421     table-align-comparator .bool_set:N =
422         \l__siunitx_table_align_comparator_bool ,
423     table-align-exponent .bool_set:N =
424         \l__siunitx_table_align_exponent_bool ,
425     table-align-text-after .bool_set:N =
426         \l__siunitx_table_align_after_bool ,
427     table-align-text-before .bool_set:N =
428         \l__siunitx_table_align_before_bool ,
429     table-align-uncertainty .bool_set:N =
430         \l__siunitx_table_align_uncertainty_bool
431 }

```

(End definition for \l__siunitx_table_align_comparator_bool and others.)

```

\__siunitx_table_print:nnn
\__siunitx_table_print:VVV
432 \cs_new_protected:Npn \__siunitx_table_print:nnn #1#2#3
    \_siunitx_table_print_marker:nnn
433 { \use:c { __siunitx_table_print_ \l__siunitx_table_align_mode_tl :nnn } {#1} {#2} {#3} }
    \_siunitx_table_print_marker:w
434 \cs_generate_variant:Nn \__siunitx_table_print:nnn { VVV }
    \_siunitx_table_print_format:nnn
    \_siunitx_table_print_marker_auxi:w
    \_siunitx_table_print_marker_auxii:w
    \_siunitx_table_print_marker_auxiii:w
    \_siunitx_table_print_format_after:N
    \_siunitx_table_print_format_box:Nn
    \_siunitx_table_print_none:nnn

```

When centering on the decimal marker, alignment is relatively simple, and close in concept to that used without parsing. First we need to deal with any text before or after the number. For text *before*, there's the case where it has no width and might be a font or color change: that has to be filtered out first. Then we can adjust the size of this material and that after the number such that they are equal. The number itself can then be formatted, splitting at the decimal marker. A bit more size adjustment, then the number itself and any text at the end can be inserted.

```

435 \cs_new_protected:Npn \__siunitx_table_print_marker:nnn #1#2#3
436 {
437   \hbox_set:Nn \l__siunitx_table_before_box {#1}
438   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { Opt }
439   {
440     \box_clear:N \l__siunitx_table_before_box
441     #1
442   }
443   \hbox_set:Nn \l__siunitx_table_after_box {#3}
444   \dim_compare:nNnTF
445     { \box_wd:N \l__siunitx_table_after_box }
446     > { \box_wd:N \l__siunitx_table_before_box }
447     {
448       \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
449       { \box_wd:N \l__siunitx_table_after_box }
450       {
451         \__siunitx_table_fil:
452         \hbox_unpack:N \l__siunitx_table_before_box
453       }
454     }
455     {
456       \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
457       { \box_wd:N \l__siunitx_table_before_box }
458       {
459         \hbox_unpack:N \l__siunitx_table_after_box
460         \__siunitx_table_fil:
461       }
462     }
463     \box_use_drop:N \l__siunitx_table_before_box
464     \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
465     \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
466     \tl_set:Nx \l__siunitx_table_tmp_tl
467       { \siunitx_number_format:NN \l__siunitx_table_tmp_tl \q_nil }
468     \exp_after:wN \__siunitx_table_print_marker:w
469       \l__siunitx_table_tmp_tl \q_stop
470     \box_use_drop:N \l__siunitx_table_after_box
471   }
472 \cs_new_protected:Npn \__siunitx_table_print_marker:w
473   #1 \q_nil #2 \q_nil #3 \q_stop
474   {
475     \hbox_set:Nn \l__siunitx_table_integer_box
476       { \siunitx_print:nn { number } { #1#2 } }
477     \hbox_set:Nn \l__siunitx_table_decimal_box
478       { \siunitx_print:nn { number } { \__siunitx_table_cleanup_decimal:w #3 } }
479     \__siunitx_table_center_marker:
480     \box_use_drop:N \l__siunitx_table_integer_box

```

```

481 \box_use_drop:N \l__siunitx_table_decimal_box
482 }

```

For positioning based on a format, we have to work part-by-part as there are a number of alignment points to get right. As for the `marker` approach, first we check if the material before the numerical content is of zero width. Next we need to format the model and content numbers, before starting an auxiliary chain to pick out the various parts in order.

```

483 \cs_new_protected:Npn \__siunitx_table_print_format:nnn #1#2#3
484 {
485   \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_before_model_tl }
486   \hbox_set:Nn \l__siunitx_table_before_box {#1}
487   \dim_compare:nNnT { \box_wd:N \l__siunitx_table_before_box } = { 0pt }
488   {
489     \box_clear:N \l__siunitx_table_before_box
490     #1
491   }
492   \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
493   { \box_wd:N \l__siunitx_table_tmp_box }
494   {
495     \__siunitx_table_fil:
496     \hbox_unpack:N \l__siunitx_table_before_box
497   }
498   \siunitx_number_parse:nN {#2} \l__siunitx_table_tmp_tl
499   \group_begin:
500     \bool_if:NT \l__siunitx_table_auto_round_bool
501     {
502       \exp_args:Nx \keys_set:nn { siunitx }
503       {
504         round-mode      = places ,
505         round-pad       = true   ,
506         round-precision =
507         \exp_after:wN \__siunitx_table_print_format:nnnnnn
508         \l__siunitx_table_format_tl
509       }
510     }
511     \siunitx_number_process:NN \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
512     \exp_args:NNNV \group_end:
513     \tl_set:Nn \l__siunitx_table_tmp_tl \l__siunitx_table_tmp_tl
514     \tl_set:Nx \l__siunitx_table_tmp_tl
515     {
516       \siunitx_number_format:NN \l__siunitx_table_model_tl \q_nil
517       \exp_not:N \q_mark
518       \siunitx_number_format:NN \l__siunitx_table_tmp_tl \q_nil
519     }
520     \exp_after:wN \__siunitx_table_print_format_auxi:w
521     \l__siunitx_table_tmp_tl \q_stop
522     \hbox_set:Nn \l__siunitx_table_tmp_box { \l__siunitx_table_after_model_tl }
523     \hbox_set_to_wd:Nnn \l__siunitx_table_after_box
524     { \box_wd:N \l__siunitx_table_tmp_box + \l__siunitx_table_carry_dim }
525     {
526       \bool_if:NT \l__siunitx_table_align_after_bool
527       { \skip_horizontal:n { \l__siunitx_table_carry_dim } }
528       #3
529       \__siunitx_table_fil:

```



```

530     }
531     \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
532     {
533         \box_use_drop:N \l__siunitx_table_before_box
534         \box_use_drop:N \l__siunitx_table_integer_box
535         \box_use_drop:N \l__siunitx_table_decimal_box
536         \box_use_drop:N \l__siunitx_table_after_box
537     }
538 }
539 \cs_new:Npn \__siunitx_table_print_format:nnnnnn #1#2#3#4#5#6#7
540 { 0 #4 }

```

The first numerical part to handle is the comparator. Any white space we need to add goes into the text part *if* alignment is not active (*i.e.* we are looking “backwards” to place this filler).

```

541 \cs_new_protected:Npn \__siunitx_table_print_format_auxi:w
542 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
543 {
544     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}
545     \bool_if:NTF \l__siunitx_table_align_before_bool
546     {
547         \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
548         { \box_wd:N \l__siunitx_table_tmp_box }
549         {
550             \__siunitx_table_fil:
551             \tl_if_blank:nF {#3}
552             { \siunitx_print:nn { number } {#3} }
553         }
554     }
555     {
556         \__siunitx_table_print_format_box:Nn \l__siunitx_table_integer_box {#3}
557         \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
558         {
559             \box_wd:N \l__siunitx_table_before_box
560             + \box_wd:N \l__siunitx_table_tmp_box
561             - \box_wd:N \l__siunitx_table_integer_box
562         }
563         {
564             \__siunitx_table_fil:
565             \hbox_unpack:N \l__siunitx_table_before_box
566         }
567     }
568     \__siunitx_table_print_format_auxii:w #2 \q_mark #4 \q_stop
569 }

```

The integer part follows much the same pattern, except now it is control of the comparator alignment that determines where the white space goes. As we already have content in the **integer** box, we need to measure how much *extra* material has been added. To avoid using more boxes or re-setting, we do that by recording sizes before and after the change. (In effect, `\l__siunitx_table_tmp_dim` is here “`\l__@@_comparator_dim`”.)

```

570 \cs_new_protected:Npn \__siunitx_table_print_format_auxii:w
571 #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
572 {
573     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1}

```

```

574 \bool_lazy_and:nnTF
575 { \l__siunitx_table_align_comparator_bool }
576 { \dim_compare_p:nNn { \box_wd:N \l__siunitx_table_integer_box } > { 0pt } }
577 {
578   \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
579   {
580     \box_wd:N \l__siunitx_table_integer_box
581     + \box_wd:N \l__siunitx_table_tmp_box
582   }
583   {
584     \hbox_unpack:N \l__siunitx_table_integer_box
585     \__siunitx_table_fil:
586     \siunitx_print:nn { number } {#3}
587   }
588 }
589 {
590   \bool_if:NTF \l__siunitx_table_align_before_bool
591   {
592     \hbox_set_to_wd:Nnn \l__siunitx_table_integer_box
593     {
594       \box_wd:N \l__siunitx_table_integer_box
595       + \box_wd:N \l__siunitx_table_tmp_box
596     }
597     {
598       \__siunitx_table_fil:
599       \hbox_unpack:N \l__siunitx_table_integer_box
600       \siunitx_print:nn { number } {#3}
601     }
602   }
603   {
604     \dim_set:Nn \l__siunitx_table_tmp_dim
605     { \box_wd:N \l__siunitx_table_integer_box }
606     \hbox_set:Nn \l__siunitx_table_integer_box
607     {
608       \hbox_unpack:N \l__siunitx_table_integer_box
609       \siunitx_print:nn { number } {#3}
610     }
611     \hbox_set_to_wd:Nnn \l__siunitx_table_before_box
612     {
613       \box_wd:N \l__siunitx_table_before_box
614       + \box_wd:N \l__siunitx_table_tmp_box
615       + \l__siunitx_table_tmp_dim
616       - \box_wd:N \l__siunitx_table_integer_box
617     }
618     {
619       \__siunitx_table_fil:
620       \hbox_unpack:N \l__siunitx_table_before_box
621     }
622   }
623 }
624 \__siunitx_table_print_format_auxiii:w #2 \q_mark #4 \q_stop
625 }

```

We now deal with the decimal part: there is nothing already in the decimal box, so the basics are easy. We need to “carry forward” any white space, as where it gets inserted

depends on the options for subsequent parts.

```

626 \cs_new_protected:Npn \__siunitx_table_print_format_auxiii:w
627   #1 \q_nil #2 \q_nil #3 \q_mark #4 \q_nil #5 \q_nil #6 \q_stop
628   {
629     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box {#1#2}
630     \__siunitx_table_print_format_box:Nn \l__siunitx_table_decimal_box {#4#5}
631     \dim_set:Nn \l__siunitx_table_carry_dim
632       {
633         \box_wd:N \l__siunitx_table_tmp_box
634         - \box_wd:N \l__siunitx_table_decimal_box
635       }
636     \__siunitx_table_print_format_auxiv:w #3 \q_mark #6 \q_stop
637   }

```

Any separated uncertainty is now picked up. That has a number of parts, so the first step is to look for a sign (which will be #1). We then split, either simply tidying up the markers if there is no uncertainty, or setting it.

```

638 \cs_new_protected:Npn \__siunitx_table_print_format_auxiv:w
639   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
640   {
641     \tl_if_blank:nTF {#1}
642       { \__siunitx_table_print_format_auxv:w }
643       { \__siunitx_table_print_format_auxvi:w }
644       #1#2 \q_mark #3#4 \q_stop
645   }
646 \cs_new_protected:Npn \__siunitx_table_print_format_auxv:w
647   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
648   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
649   { \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop }

```

Sorting out the placement of the uncertainty requires both the model and real data widths, so we store the former to avoiding needing more boxes. It's then just a case of putting the carry-over white space in the right place.

```

650 \cs_new_protected:Npn \__siunitx_table_print_format_auxvi:w
651   #1 \q_nil #2 \q_nil #3 \q_nil #4 \q_mark
652   #5 \q_nil #6 \q_nil #7 \q_nil #8 \q_stop
653   {
654     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2#3 }
655     \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }
656     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #5#6#7 }
657     \__siunitx_table_print_format_after:N \l__siunitx_table_align_uncertainty_bool
658     \__siunitx_table_print_format_auxvii:w #4 \q_mark #8 \q_stop
659   }

```

Finally, we get to the exponent part: the multiplication symbol is #1 and the number itself is #2. The code is almost the same as for uncertainties, which allows a shared auxiliary to be used.

```

660 \cs_new_protected:Npn \__siunitx_table_print_format_auxvii:w
661   #1 \q_nil #2 \q_mark #3 \q_nil #4 \q_stop
662   {
663     \tl_if_blank:nF {#2}
664     {
665       \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #1#2 }
666       \dim_set:Nn \l__siunitx_table_tmp_dim { \box_wd:N \l__siunitx_table_tmp_box }

```

```

667     \__siunitx_table_print_format_box:Nn \l__siunitx_table_tmp_box { { } #3#4 }
668     \__siunitx_table_print_format_after:N \l__siunitx_table_align_exponent_bool
669   }
670 }

```

A simple auxiliary to avoid relatively expensive use of the print routine for empty parts.

```

671 \cs_new_protected:Npn \__siunitx_table_print_format_box:Nn #1#2
672 {
673   \hbox_set:Nn #1
674   {
675     \tl_if_blank:nF {#2}
676     { \siunitx_print:nn { number } {#2} }
677   }
678 }

```

A common routine for placing material after the decimal marker and “shuffling”.

```

679 \cs_new_protected:Npn \__siunitx_table_print_format_after:N #1
680 {
681   \bool_if:NTF #1
682   {
683     \hbox_set_to_wd:Nnn \l__siunitx_table_decimal_box
684     {
685       \box_wd:N \l__siunitx_table_decimal_box
686       + \l__siunitx_table_carry_dim
687       + \box_wd:N \l__siunitx_table_tmp_box
688     }
689     {
690       \hbox_unpack:N \l__siunitx_table_decimal_box
691       \__siunitx_table_fil:
692       \hbox_unpack:N \l__siunitx_table_tmp_box
693     }
694     \dim_set:Nn \l__siunitx_table_carry_dim
695     {
696       \l__siunitx_table_tmp_dim
697       - \box_wd:N \l__siunitx_table_tmp_box
698     }
699   }
700   {
701     \hbox_set:Nn \l__siunitx_table_decimal_box
702     {
703       \hbox_unpack:N \l__siunitx_table_decimal_box
704       \hbox_unpack:N \l__siunitx_table_tmp_box
705     }
706     \dim_add:Nn \l__siunitx_table_carry_dim
707     {
708       \l__siunitx_table_tmp_dim
709       - \box_wd:N \l__siunitx_table_tmp_box
710     }
711   }
712 }

```

With no alignment, everything supplied is treated more-or-less the same as `\num` (but without the `xparse` wrapper).

```

713 \cs_new_protected:Npn \__siunitx_table_print_none:nnn #1#2#3
714 {

```

```

715 \use:c { __siunitx_table_align_ \l__siunitx_table_align_number_tl :n }
716 {
717   #1
718   \siunitx_number_format:nN {#2} \l__siunitx_table_tmp_tl
719   \siunitx_print:nV { number } \l__siunitx_table_tmp_tl
720   #3
721 }
722 }

```

(End definition for `_siunitx_table_print:nnn` and others.)

2.9 Standard settings for module options

Some of these follow naturally from the point of definition (*e.g.* boolean variables are always `false` to begin with), but for clarity everything is set here.

```

723 \keys_set:nn { siunitx }
724 {
725   table-align-comparator = true   ,
726   table-align-exponent   = true   ,
727   table-align-text-after  = true   ,
728   table-align-text-before = true   ,
729   table-align-uncertainty = true   ,
730   table-alignment         = center ,
731   table-auto-round        = false  ,
732   table-column-width      = 0pt    ,
733   table-fixed-width       = false  ,
734   table-format            = 2.2    ,
735   table-number-alignment  = center ,
736   table-text-alignment    = center ,

```

Out of order as `table-format` sets this implicitly too.

```

737   table-alignment-mode    = marker
738 }
739 \endpackage

```

Part IX

siunitx-unit – Parsing and formatting units

This submodule is dedicated to formatting physical units. The main function, `\siunitx-unit_format:nN`, takes user input specify physical units and converts it into a formatted token list suitable for typesetting in math mode. While the formatter will deal correctly with “literal” user input, the key strength of the module is providing a method to describe physical units in a “symbolic” manner. The output format of these symbolic units can then be controlled by a number of key–value options made available by the module.

A small number of L^AT_EX 2_ε math mode commands are assumed to be available as part of the formatted output. The `\mathchoice` command (normally the T_EX primitive) is needed when using `per-mode = symbol-or-fraction`. The commands `\frac`, `\mathrm`, `\mbox`, `\sq` and `\,`, are used by the standard module settings, and `\ensuremath`, `\hbar`, `\mathit` and `\mathrm` in some standard unit definitions (for atomic and natural units). For the display of colored (highlighted) and cancelled units, the commands `\textcolor` and `\cancel` are assumed to be available.

1 Formatting units

```
\siunitx_unit_format:nN
\siunitx_unit_format:xN
```

```
\siunitx_unit_format:nN {<units>} <tl var>
```

This function converts the input `<units>` into a processed `<tl var>` which can then be inserted in math mode to typeset the material. Where the `<units>` are given in symbolic form, described elsewhere, this formatting process takes place in two stages: the `<units>` are parsed into a structured form before the generation of the appropriate output form based on the active settings. When the `<units>` are given as literals, processing is minimal: the characters `.` and `~` are converted to unit products (boundaries). In both cases, the result is a series of tokens intended to be typeset in math mode with appropriate choice of font for typesetting of the textual parts.

For example,

```
\siunitx_unit_format:nN { \kilo \metre \per \second } \l_tmpa_tl
```

will, with standard settings, result in `\l_tmpa_tl` being set to

```
\mathrm{km}\,,\mathrm{s}^{-1}
```

<code>\siunitx_unit_format:nNN</code>	<code>\siunitx_unit_format:nNN {<units>} <tl var> <fp var></code>
---------------------------------------	---

This function formats the $\langle units \rangle$ in the same way as described for `\siunitx_unit_format:nN`. When the input is given in symbolic form, any decimal unit prefixes will be extracted and the overall power of ten that these represent will be stored in the $\langle fp var \rangle$.

For example,

```
\siunitx_unit_format:nNN { \kilo \metre \per \second }
\l_tmpa_tl \l_tmpa_fp
```

will, with standard settings, result in `\l_tmpa_tl` being set to

```
\mathrm{m}\,,\mathrm{s}^{-1}
```

with `\l_tmpa_fp` taking value 3. Note that the latter is a floating point variable: it is possible for non-integer values to be obtained here.

2 Defining symbolic units

<code>\siunitx_declare_prefix:Nnn</code>	<code>\siunitx_declare_prefix:Nnn <prefix> {<power>} {<symbol>}</code>
<code>\siunitx_declare_prefix:Nnx</code>	

Defines a symbolic $\langle prefix \rangle$ (which should be a control sequence such as `\kilo`) to be converted by the parser to the $\langle symbol \rangle$. The latter should consist of literal content (e.g. `k`). In literal mode the $\langle symbol \rangle$ will be typeset directly. The prefix should represent an integer $\langle power \rangle$ of 10, and this information may be used to convert from one or more $\langle prefix \rangle$ symbols to an overall power applying to a unit. See also `\siunitx_declare_prefix:Nn`.

<code>\siunitx_declare_prefix:Nn</code>	<code>\siunitx_declare_prefix:Nn <prefix> {<symbol>}</code>
---	---

Defines a symbolic $\langle prefix \rangle$ (which should be a control sequence such as `\kilo`) to be converted by the parser to the $\langle symbol \rangle$. The latter should consist of literal content (e.g. `k`). In literal mode the $\langle symbol \rangle$ will be typeset directly. In contrast to `\siunitx_declare_prefix:Nnn`, there is no assumption about the mathematical nature of the $\langle prefix \rangle$, i.e. the prefix may represent a power of any base. As a result, no conversion of the $\langle prefix \rangle$ to a numerical power will be possible.

<code>\siunitx_declare_power:NnN</code>	<code>\siunitx_declare_power:NnN <pre-power> <post-power> {<value>}</code>
---	--

Defines *two* symbolic $\langle powers \rangle$ (which should be control sequences such as `\squared`) to be converted by the parser to the $\langle value \rangle$. The latter should be an integer or floating point number in the format defined for `l3fp`. Powers may precede a unit or be give after it: both forms are declared at once, as indicated by the argument naming. In literal mode, the $\langle value \rangle$ will be applied as a superscript to either the next token in the input (for the $\langle pre-power \rangle$) or appended to the previously-typeset material (for the $\langle post-power \rangle$).

<code>\siunitx_declare_qualifier:Nn</code>	<code>\siunitx_declare_qualifier:Nn <qualifier> {<meaning>}</code>
--	--

Defines a symbolic $\langle qualifier \rangle$ (which should be a control sequence such as `\catalyst`) to be converted by the parser to the $\langle meaning \rangle$. The latter should consist of literal content (e.g. `cat`). In literal mode the $\langle meaning \rangle$ will be typeset following a space after the unit to which it applies.

<hr/> <code>\siunitx_declare_unit:Nn</code> <hr/>	<code>\siunitx_declare_unit:Nn <unit> {<meaning>}</code>
<code>\siunitx_declare_unit:Nx</code>	<code>\siunitx_declare_unit:Nnn <unit> {<meaning>} {<options>}</code>
<code>\siunitx_declare_unit:Nnn</code> <hr/>	Defines a symbolic $\langle unit \rangle$ (which should be a control sequence such as <code>\kilogram</code>) to be converted by the parser to the $\langle meaning \rangle$. The latter may consist of literal content (e.g. <code>kg</code>), other symbolic unit commands (e.g. <code>\kilo\gram</code>) or a mixture of the two. In literal mode the $\langle meaning \rangle$ will be typeset directly. The version taking an $\langle options \rangle$ argument may be used to support per-unit options: these are applied at the top level or using <code>\siunitx_unit_options_apply:n</code> .

<hr/> <code>\l_siunitx_unit_font_tl</code> <hr/>	The font function which is applied to the text of units when constructing formatted units: set by <code>font-command</code> .
--	---

<hr/> <code>\l_siunitx_unit_symbolic_seq</code> <hr/>	This sequence contains all of the symbolic names defined: these will be in the form of control sequences such as <code>\kilogram</code> . The order of the sequence is unimportant. This includes prefixes and powers as well as units themselves.
---	--

<hr/> <code>\l_siunitx_unit_seq</code> <hr/>	This sequence contains all of the symbolic <i>unit</i> names defined: these will be in the form of control sequences such as <code>\kilogram</code> . In contrast to <code>\l_siunitx_unit_symbolic_seq</code> , it <i>only</i> holds units themselves
--	--

3 Per-unit options

<hr/> <code>\siunitx_unit_options_apply:n</code> <hr/>	<code>\siunitx_unit_options_apply:n <unit(s)></code> Applies any unit-specific options set up using <code>\siunitx_declare_unit:Nnn</code> . This allows there use outside of unit formatting, for example to influence spacing in quantities. The options are applied only once at a given group level, which allows for user over-ride <i>via</i> <code>\keys_set:nn { siunitx } { ... }</code> .
--	--

4 Units in (PDF) strings

<hr/> <code>\siunitx_unit_pdfstring_context:</code> <hr/>	<code>\group_begin:</code> <code>\siunitx_unit_pdfstring_context:</code> <code><Expansion context> <units></code> <code>\group_end:</code> Sets symbol unit macros to generate text directly. This is needed in expansion contexts where units must be converted to simple text. This function is itself not expandable, so must be using within a surrounding group as show in the example.
---	--

5 Pre-defined symbolic unit components

The unit parser is defined to recognise a number of pre-defined units, prefixes and powers, and also interpret a small selection of “generic” symbolic parts.

Broadly, the pre-defined units are those defined by the BIPM in the documentation for the *International System of Units* (SI) [1]. As far as possible, the names given to the command names for units are those used by the BIPM, omitting spaces and using only ASCII characters. The standard symbols are also taken from the same documentation. In the following documentation, the order of the description of units broadly follows the SI Brochure.

<code>\kilogram</code>	The base units as defined in Section 2.1 of the SI Brochure [2]. Notice that <code>\meter</code> is defined as an alias for <code>\metre</code> as the former spelling is common in the US (although the latter is the official spelling).
<code>\metre</code>	
<code>\meter</code>	
<code>\mole</code>	
<code>\kelvin</code>	
<code>\candela</code>	
<code>\second</code>	
<code>\ampere</code>	

<code>\gram</code>	The base unit <code>\kilogram</code> is defined using an SI prefix: as such the (derived) unit <code>\gram</code> is required by the module to correctly produce output for the <code>\kilogram</code> .
--------------------	--

<code>\yocto</code>	Prefixes, all of which are integer powers of 10: the powers are stored internally by the module and can be used for conversion from prefixes to their numerical equivalent. These prefixes are documented in Section 3.1 of the SI Brochure [4].
<code>\zepto</code>	
<code>\atto</code>	Note that the <code>\kilo</code> prefix is required to define the base <code>\kilogram</code> unit. Also note the two spellings available for <code>\deca</code> / <code>\deka</code> .
<code>\femto</code>	
<code>\pico</code>	
<code>\nano</code>	
<code>\micro</code>	
<code>\milli</code>	
<code>\centi</code>	
<code>\deci</code>	
<code>\deca</code>	
<code>\deka</code>	
<code>\hecto</code>	
<code>\kilo</code>	
<code>\mega</code>	
<code>\giga</code>	
<code>\tera</code>	
<code>\peta</code>	
<code>\exa</code>	
<code>\zetta</code>	
<code>\yotta</code>	

<hr/>	
<code>\becquerel</code>	The defined SI units with defined names and symbols, as given in Section 2.2.2 of the SI Brochure [3]. Notice that the names of the units are lower case with the exception of <code>\degreeCelsius</code> , and that this unit name includes “degree”.
<code>\degreeCelsius</code>	
<code>\coulomb</code>	
<code>\farad</code>	
<code>\gray</code>	
<code>\hertz</code>	
<code>\henry</code>	
<code>\joule</code>	
<code>\katal</code>	
<code>\lumen</code>	
<code>\lux</code>	
<code>\newton</code>	
<code>\ohm</code>	
<code>\pascal</code>	
<code>\radian</code>	
<code>\siemens</code>	
<code>\sievert</code>	
<code>\steradian</code>	
<code>\tesla</code>	
<code>\volt</code>	
<code>\watt</code>	
<code>\weber</code>	
<hr/>	

<hr/>	
<code>\day</code>	Units accepted for use with the SI: here <code>\minute</code> is a unit of time not of plane angle.
<code>\hectare</code>	
<code>\hour</code>	These units are taken from Table 4.1 of the SI Brochure [6]. For the unit <code>\litre</code> , both <code>l</code> and <code>L</code> are listed as acceptable symbols: the latter is the standard setting of the module. The alternative spelling <code>\liter</code> is also given for this unit for US users (as with <code>\metre</code> , the official spelling is “re”).
<code>\litre</code>	
<code>\liter</code>	
<code>\minute</code>	
<code>\tonne</code>	
<hr/>	

<hr/>	
<code>\arcminute</code>	Units for plane angles accepted for use with the SI: to avoid a clash with units for time, here <code>\arcminute</code> and <code>\arcsecond</code> are used in place of <code>\minute</code> and <code>\second</code> . These units are taken from Table 4.1 of the SI Brochure [6].
<code>\arcsecond</code>	
<code>\degree</code>	
<hr/>	

<hr/> <code>\astronomicalunit</code> <code>\atomicmassunit</code> <code>\auction</code> <code>\auctharge</code> <code>\aucthenergy</code> <code>\aucthlength</code> <code>\aucthmass</code> <code>\aucthime</code> <code>\bohr</code> <code>\dalton</code> <code>\electronvolt</code> <code>\hartree</code> <code>\nuaction</code> <code>\numass</code> <code>\nuspeed</code> <code>\nutime</code> <hr/>	<p>Non-SI where values must be determined experimentally. These units are taken from Table 7 of the SI Brochure [7]. Where no better name is given for the unit in the SI Brochure, the prefixes <code>nu</code> (natural unit) and <code>au</code> (atomic unit) are used.</p> <p>Note that the value of the natural unit of speed (the speed of light) is used to define the second and is thus not determined by experiment: it is however included in this set of units.</p>
<hr/> <code>\angstrom</code> <code>\bar</code> <code>\barn</code> <code>\bel</code> <code>\decibel</code> <code>\knot</code> <code>\millimetremercury</code> <code>\nauticalmile</code> <code>\neper</code> <hr/>	<p>Non-SI units accepted for use with the SI. These units are taken from Table 8 of the SI Brochure [8].</p>
<hr/> <code>\dyne</code> <code>\erg</code> <code>\gal</code> <code>\gauss</code> <code>\maxwell</code> <code>\oersted</code> <code>\phot</code> <code>\poise</code> <code>\stilb</code> <code>\stokes</code> <hr/>	<p>Non-SI units associated with the CGS and the CGS-Gaussian system of units. These units are taken from Table 9 of the SI Brochure [9].</p>
<hr/> <code>\percent</code> <hr/>	<p>The mathematical concept of percent, usable with the SI as detailed in Section 5.3.7 of the SI Brochure [5].</p>
<hr/> <code>\square</code> <code>\cubic</code> <hr/>	<p><code>\square</code> $\langle prefix \rangle \langle unit \rangle$ <code>\cubic</code> $\langle prefix \rangle \langle unit \rangle$</p>
	<p>Pre-defined unit powers which apply to the next $\langle prefix \rangle / \langle unit \rangle$ combination.</p>
<hr/> <code>\squared</code> <code>\cubed</code> <hr/>	<p>$\langle prefix \rangle \langle unit \rangle$ <code>\squared</code> $\langle prefix \rangle \langle unit \rangle$ <code>\cubed</code></p>
	<p>Pre-defined unit powers which apply to the preceding $\langle prefix \rangle / \langle unit \rangle$ combination.</p>

<hr/> \per <hr/>	\per $\langle prefix \rangle$ $\langle unit \rangle$ $\langle power \rangle$
	Indicates that the next $\langle prefix \rangle/\langle unit \rangle/\langle power \rangle$ combination is reciprocal, <i>i.e.</i> raises it to the power -1 . This symbolic representation may be applied in addition to a \power , and will work correctly if the \power itself is negative. In literal mode \per will print a slash (“/”).
<hr/> \cancel <hr/>	\cancel $\langle prefix \rangle$ $\langle unit \rangle$ $\langle power \rangle$
	Indicates that the next $\langle prefix \rangle/\langle unit \rangle/\langle power \rangle$ combination should be “cancelled out”. In the parsed output, the entire unit combination will be given as the argument to a function \cancel , which is assumed to be available at a higher level. In literal mode, the same higher-level \cancel will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for \cancel outside of the scope of the unit parser.
<hr/> \highlight <hr/>	\highlight $\{ \langle color \rangle \}$ $\langle prefix \rangle$ $\langle unit \rangle$ $\langle power \rangle$
	Indicates that the next $\langle prefix \rangle/\langle unit \rangle/\langle power \rangle$ combination should be highlighted in the specified $\langle color \rangle$. In the parsed output, the entire unit combination will be given as the argument to a function \textcolor , which is assumed to be available at a higher level. In literal mode, the same higher-level \textcolor will be applied to the next token. It is the responsibility of the calling code to provide an appropriate definition for \textcolor outside of the scope of the unit parser.
<hr/> \of <hr/>	$\langle prefix \rangle$ $\langle unit \rangle$ $\langle power \rangle$ \of $\{ \langle qualifier \rangle \}$
	Indicates that the $\langle qualifier \rangle$ applies to the current $\langle prefix \rangle/\langle unit \rangle/\langle power \rangle$ combination. In parsed mode, the display of the result will depend upon module options. In literal mode, the $\langle qualifier \rangle$ will be printed in parentheses following the preceding $\langle unit \rangle$ and a full-width space.
<hr/> \raiseto \tothe <hr/>	\raiseto $\{ \langle power \rangle \}$ $\langle prefix \rangle$ $\langle unit \rangle$ $\langle prefix \rangle$ $\langle unit \rangle$ \tothe $\{ \langle power \rangle \}$
	Indicates that the $\langle power \rangle$ applies to the current $\langle prefix \rangle/\langle unit \rangle$ combination. As shown, \raiseto applies to the next $\langle unit \rangle$ whereas \tothe applies to the preceding unit. In literal mode the \power will be printed as a superscript attached to the next token (\raiseto) or preceding token (\tothe) as appropriate.

5.1 Key-value options

The options defined by this submodule are available within the l3keys **siunitx** tree.

<hr/> bracket-unit-denominator <hr/>	bracket-unit-denominator = true false
	Switch to determine whether brackets are added to the denominator part of a unit when printed using inline fractional form (with per-mode as repeated-symbol , symbol or symbol-or-fraction). The standard setting is true .
<hr/> forbid-literal-units <hr/>	forbid-literal-units = true false
	Switch which determines if literal units are allowed when parsing is active; does not apply when parse-units is false .

<u>font-command</u>	<code>font-command = $\langle command \rangle$</code> <p>Command applied to text during output of units: should be command usable in math mode for font selection. Notice that in a typical unit this does not (necessarily) apply to all output, for example powers or brackets. The standard setting is <code>\mathrm</code>.</p>
<u>fraction-command</u>	<code>fraction-command = $\langle command \rangle$</code> <p>Command used to create fractional output when <code>per-mode</code> is set to <code>fraction</code>. The standard setting is <code>\frac</code>.</p>
<u>inter-unit-product</u>	<code>inter-unit-product = $\langle separator \rangle$</code> <p>Inserted between unit combinations in parsed mode, and used to replace <code>.</code> and <code>~</code> in literal mode. The standard setting is <code>\,</code>.</p>
<u>parse-units</u>	<code>parse-units = true false</code> <p>Determines whether parsing of unit symbols is attempted or literal mode is used directly. The standard setting is <code>true</code>.</p>
<u>per-mode</u>	<code>per-mode = fraction power power-positive-first repeated-symbol symbol symbol-or-fraction</code> <p>Selects how the negative powers (<code>\per</code>) are formatted: a choice from the options <code>fraction</code>, <code>power</code>, <code>power-positive-first</code>, <code>repeated-symbol</code>, <code>symbol</code> and <code>symbol-or-fraction</code>. The option <code>fraction</code> generates fractional output when appropriate using the command specified by the <code>fraction-command</code> option. The setting <code>power</code> uses reciprocal powers leaving the units in the order of input, while <code>power-positive-first</code> uses the same display format but sorts units such that the positive powers come before negative ones. The <code>symbol</code> setting uses a symbol (specified by <code>per-symbol</code>) between positive and negative powers, while <code>repeated-symbol</code> uses the same symbol but places it before <i>every</i> unit with a negative power (this is mathematically “wrong” but often seen in real work). Finally, <code>symbol-or-fraction</code> acts like <code>symbol</code> for inline output and like <code>fraction</code> when the output is used in a display math environment. The standard setting is <code>power</code>.</p>
<u>per-symbol</u>	<code>per-symbol = $\langle symbol \rangle$</code> <p>Specifies the symbol to be used to denote negative powers when the option <code>per-mode</code> is set to <code>repeated-symbol</code>, <code>symbol</code> or <code>symbol-or-fraction</code>. The standard setting is <code>/</code>.</p>
<u>qualifier-mode</u>	<code>qualifier-mode = bracket combine phrase subscript</code> <p>Selects how qualifiers are formatted: a choice from the options <code>bracket</code>, <code>combine</code>, <code>phrase</code> and <code>subscript</code>. The option <code>bracket</code> wraps the qualifier in parenthesis, <code>combine</code> joins the qualifier with the unit directly, <code>phrase</code> joins the material using <code>qualifier-phrase</code> as a link, and <code>subscript</code> formats the qualifier as a subscript. The standard setting is <code>subscript</code>.</p>
<u>qualifier-phrase</u>	<code>qualifier-phrase = $\langle phrase \rangle$</code> <p>Defines the $\langle phrase \rangle$ used when <code>qualifier-mode</code> is set to <code>phrase</code>.</p>

sticky-per	<code>sticky-per = true false</code>
-------------------	--------------------------------------

Used to determine whether `\per` should be applied one a unit-by-unit basis (when `false`) or should apply to all following units (when `true`). The latter mode is somewhat akin conceptually to the \TeX `\over` primitive. The standard setting is `false`.

unit-close-bracket	<code>unit-close-bracket = $\langle symbol \rangle$</code>
---------------------------	---

Bracket symbol used to close a matched pair around units when once is required to maintain mathematical logic. The standard setting is `)`.

unit-open-bracket	<code>unit-open-bracket = $\langle symbol \rangle$</code>
--------------------------	--

Bracket symbol used to open a matched pair around units when once is required to maintain mathematical logic. The standard setting is `(`.

6 siunitx-unit implementation

Start the DocStrip guards.

```
1  $\langle *package \rangle$ 
```

Identify the internal prefix (\LaTeX 3 DocStrip convention): only internal material in this *submodule* should be used directly.

```
2  $\langle @@=siunitx\_unit \rangle$ 
```

6.1 Initial set up

The mechanisms defined here need a few variables to exist and to be correctly set: these don't belong to one subsection and so are created in a small general block.

Variants not provided by `expl3`.

```
3 \cs_generate_variant:Nn \tl_replace_all:Nnn { NnV }
```

<code>\l__siunitx_unit_tmp_fp</code>	Scratch space.
--------------------------------------	----------------

<code>\l__siunitx_unit_tmp_int</code>	4 <code>\fp_new:N \l__siunitx_unit_tmp_fp</code>
<code>\l__siunitx_unit_tmp_tl</code>	5 <code>\int_new:N \l__siunitx_unit_tmp_int</code>
	6 <code>\tl_new:N \l__siunitx_unit_tmp_tl</code>

(End definition for `\l__siunitx_unit_tmp_fp`, `\l__siunitx_unit_tmp_int`, and `\l__siunitx_unit_tmp_tl`.)

<code>\c_siunitx_unit_math_subscript_tl</code>	Useful tokens with awkward category codes.
--	--

```
7 \tl_const:Nx \c_siunitx_unit_math_subscript_tl
8 { \char_generate:nn { '\_ } { 8 } }
```

(End definition for `\c_siunitx_unit_math_subscript_tl`.)

<code>\l__siunitx_unit_parsing_bool</code>	A boolean is used to indicate when the symbolic unit functions should produce symbolic or literal output. This is used when the symbolic names are used along with literal input, and ensures that there is a sensible fall-back for these cases.
--	---

```
9 \bool_new:N \l__siunitx_unit_parsing_bool
```

(End definition for `\l__siunitx_unit_parsing_bool`.)

`\l__siunitx_unit_test_bool` A switch used to indicate that the code is testing the input to find if there is any typeset output from individual unit macros. This is needed to allow the “base” macros to be found, and also to pick up the difference between symbolic and literal unit input.

```
10 \bool_new:N \l__siunitx_unit_test_bool
```

(End definition for `\l__siunitx_unit_test_bool`.)

`_siunitx_unit_if_symbolic:nTF` The test for symbolic units is needed in two places. First, there is the case of “pre-parsing” input to check if it can be parsed. Second, when parsing there is a need to check if the current unit is built up from others (symbolic) or is defined in terms of some literals. To do this, the approach used is to set all of the symbolic unit commands expandable and to do nothing, with the few special cases handled manually.

```
11 \prg_new_protected_conditional:Npnn \_siunitx_unit_if_symbolic:n #1 { TF }
12 {
13   \group_begin:
14     \bool_set_true:N \l__siunitx_unit_test_bool
15     \protected@edef \l__siunitx_unit_tmp_tl {#1}
16     \exp_args:NNV \group_end:
17     \tl_if_blank:nTF \l__siunitx_unit_tmp_tl
18       { \prg_return_true: }
19       { \prg_return_false: }
20 }
```

(End definition for `_siunitx_unit_if_symbolic:nTF`.)

6.2 Defining symbolic unit

Unit macros and related support are created here. These exist only within the scope of the unit processor code, thus not polluting document-level namespace and allowing overlap with other areas in the case of useful short names (for example `\pm`). Setting up the mechanisms to allow this requires a few additional steps on top of simply saving the data given by the user in creating the unit.

`\l_siunitx_unit_symbolic_seq` A list of all of the symbolic units, *etc.*, set up. This is needed to allow the symbolic names to be defined within the scope of the unit parser but not elsewhere using simple mappings.

```
21 \seq_new:N \l_siunitx_unit_symbolic_seq
```

(End definition for `\l_siunitx_unit_symbolic_seq`. This variable is documented on page 108.)

`\l_siunitx_unit_seq` A second list featuring only the units themselves.

```
22 \seq_new:N \l_siunitx_unit_seq
```

(End definition for `\l_siunitx_unit_seq`. This variable is documented on page 108.)

`_siunitx_unit_set_symbolic:Nnn`
`_siunitx_unit_set_symbolic:Npnn`
`_siunitx_unit_set_symbolic:Nnnn` The majority of the work for saving each symbolic definition is the same irrespective of the item being defined (unit, prefix, power, qualifier). This is therefore all carried out in a single internal function which does the common tasks. The three arguments here are the symbolic macro name, the literal output and the code to insert when doing full unit parsing. To allow for the “special cases” (where arguments are required) the entire mechanism is set up in a two-part fashion allowing for flexibility at the slight cost of additional functions.

Importantly, notice that the unit macros are declared as expandable. This is required so that literals can be correctly converted into a token list of material which does not depend on local redefinitions for the unit macros. That is required so that the unit formatting system can be grouped.

```

23 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Nnn #1
24   { \__siunitx_unit_set_symbolic:Nnnn #1 { } }
25 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Npnn #1#2#
26   { \__siunitx_unit_set_symbolic:Nnnn #1 {#2} }
27 \cs_new_protected:Npn \__siunitx_unit_set_symbolic:Nnnn #1#2#3#4
28   {
29     \seq_put_right:Nn \l_siunitx_unit_symbolic_seq {#1}
30     \cs_set:cpn { \__siunitx_unit_ \token_to_str:N #1 :w } #2
31     {
32       \bool_if:NF \l_siunitx_unit_test_bool
33       {
34         \bool_if:NTF \l_siunitx_unit_parsing_bool
35         {#4}
36         {#3}
37       }
38     }
39   }

```

(End definition for `__siunitx_unit_set_symbolic:Nnn`, `__siunitx_unit_set_symbolic:Npnn`, and `__siunitx_unit_set_symbolic:Nnnn`.)

`\siunitx_declare_power:NNn` Powers can come either before or after the unit. As they always come (logically) in matching, we handle this by declaring two commands, and setting each up separately.

```

40 \cs_new_protected:Npn \siunitx_declare_power:NNn #1#2#3
41   {
42     \__siunitx_unit_set_symbolic:Nnn #1
43     { \__siunitx_unit_literal_power:nn {#3} }
44     { \__siunitx_unit_parse_power:nnN {#1} {#3} \c_true_bool }
45     \__siunitx_unit_set_symbolic:Nnn #2
46     { ~ {#3} }
47     { \__siunitx_unit_parse_power:nnN {#2} {#3} \c_false_bool }
48   }

```

(End definition for `\siunitx_declare_power:NNn`. This function is documented on page 107.)

`\siunitx_declare_prefix:Nn` For prefixes there are a couple of options. In all cases, the basic requirement is to set up to parse the prefix using the appropriate internal function. For prefixes which are powers of 10, there is also the need to be able to do conversion to/from the numerical equivalent. That is handled using two properly lists which can be used to supply the conversion data later.

`\siunitx_declare_prefix:Nnn`

`\siunitx_declare_prefix:Nnx`

```

49 \cs_new_protected:Npn \siunitx_declare_prefix:Nn #1#2
50   {
51     \__siunitx_unit_set_symbolic:Nnn #1
52     {#2}
53     { \__siunitx_unit_parse_prefix:Nn #1 {#2} }
54   }
55 \cs_new_protected:Npn \siunitx_declare_prefix:Nnn #1#2#3
56   {
57     \siunitx_declare_prefix:Nn #1 {#3}
58     \prop_put:Nnn \l_siunitx_unit_prefixes_forward_prop {#3} {#2}

```



```

59     \prop_put:Nnn \l__siunitx_unit_prefixes_reverse_prop {#2} {#3}
60   }
61   \cs_generate_variant:Nn \siunitx_declare_prefix:Nnn { Nnx }
62   \prop_new:N \l__siunitx_unit_prefixes_forward_prop
63   \prop_new:N \l__siunitx_unit_prefixes_reverse_prop

```

(End definition for `\siunitx_declare_prefix:Nn` and others. These functions are documented on page 107.)

`\siunitx_declare_qualifier:Nn` Qualifiers are relatively easy to handle: nothing to do other than save the input appropriately.

```

64   \cs_new_protected:Npn \siunitx_declare_qualifier:Nn #1#2
65   {
66     \__siunitx_unit_set_symbolic:Nnn #1
67     { ~ ( #2 ) }
68     { \__siunitx_unit_parse_qualifier:nn {#1} {#2} }
69   }

```

(End definition for `\siunitx_declare_qualifier:Nn`. This function is documented on page 107.)

`\siunitx_declare_unit:Nn` For the unit parsing, allowing for variations in definition order requires that a test is made for the output of each unit at point of use.

```

\siunitx_declare_unit:Nx
\siunitx_declare_unit:Nnn
20   \cs_new_protected:Npn \siunitx_declare_unit:Nn #1#2
21   { \siunitx_declare_unit:Nnn #1 {#2} { } }
22   \cs_generate_variant:Nn \siunitx_declare_unit:Nn { Nnx }
23   \cs_new_protected:Npn \siunitx_declare_unit:Nnn #1#2#3
24   {
25     \seq_put_right:Nn \l_siunitx_unit_seq {#1}
26     \__siunitx_unit_set_symbolic:Nnn #1
27     {#2}
28     {
29       \__siunitx_unit_if_symbolic:nTF {#2}
30       {#2}
31       { \__siunitx_unit_parse_unit:Nn #1 {#2} }
32     }
33     \tl_clear_new:c { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
34     \tl_if_empty:nF {#3}
35     { \tl_set:cn { l__siunitx_unit_options_ \token_to_str:N #1 _tl } {#3} }
36   }

```

(End definition for `\siunitx_declare_unit:Nn` and `\siunitx_declare_unit:Nnn`. These functions are documented on page 108.)

6.3 Applying unit options

`\l_siunitx_unit_options_bool`

```

87   \bool_new:N \l__siunitx_unit_options_bool

```

(End definition for `\l__siunitx_unit_options_bool`.)

`\siunitx_unit_options_apply:n` Options apply only if they have not already been set at this group level.

```

88   \cs_new_protected:Npn \siunitx_unit_options_apply:n #1
89   {
90     \bool_if:NF \l__siunitx_unit_options_bool
91     {

```

```

92     \tl_if_single_token:nT {#1}
93     {
94         \tl_if_exist:cT { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
95         {
96             \keys_set:nv { siunitx }
97             { l__siunitx_unit_options_ \token_to_str:N #1 _tl }
98         }
99     }
100 }
101 \bool_set_true:N \l__siunitx_unit_options_bool
102 }

```

(End definition for `\siunitx_unit_options_apply:n`. This function is documented on page 108.)

6.4 Non-standard symbolic units

A few of the symbolic units require non-standard definitions: these are created here. They all use parts of the more general code but have particular requirements which can only be addressed by hand. Some of these could in principle be used in place of the dedicated definitions above, but at point of use that would then require additional expansions for each unit parsed: as the macro names would still be needed, this does not offer any real benefits.

\per The `\per` symbolic unit is a bit special: it has a mechanism entirely different from everything else, so has to be set up by hand. In literal mode it is represented by a very simple symbol!

```

103 \__siunitx_unit_set_symbolic:Nnn \per
104 { / }
105 { \__siunitx_unit_parse_per: }

```

(End definition for `\per`. This function is documented on page 112.)

\cancel The two special cases, `\cancel` and `\highlight`, are easy to deal with when parsing.
\highlight When not parsing, a precaution is taken to ensure that the user level equivalents always get a braced argument.

```

106 \__siunitx_unit_set_symbolic:Npnn \cancel
107 { \__siunitx_unit_literal_special:nN { \cancel } }
108 { \__siunitx_unit_parse_special:n { \cancel } }
109 \__siunitx_unit_set_symbolic:Npnn \highlight #1
110 { \__siunitx_unit_literal_special:nN { \textcolor {#1} } }
111 { \__siunitx_unit_parse_special:n { \textcolor {#1} } }

```

(End definition for `\cancel` and `\highlight`. These functions are documented on page 112.)

\of The generic qualifier is simply the same as the dedicated ones except for needing to grab an argument.

```

112 \__siunitx_unit_set_symbolic:Npnn \of #1
113 { \ ( #1 ) }
114 { \__siunitx_unit_parse_qualifier:nn { \of {#1} } {#1} }

```

(End definition for `\of`. This function is documented on page 112.)

`\raiseto` Generic versions of the pre-defined power macros. These require an argument and so cannot be handled using the general approach. Other than that, the code here is very similar to that in `\siunitx_unit_power_set:NnN`.

```

115 \__siunitx_unit_set_symbolic:Npnn \raiseto #1
116 { \__siunitx_unit_literal_power:nn {#1} }
117 { \__siunitx_unit_parse_power:nnN { \raiseto {#1} } {#1} \c_true_bool }
118 \__siunitx_unit_set_symbolic:Npnn \tothe #1
119 { ^ {#1} }
120 { \__siunitx_unit_parse_power:nnN { \tothe {#1} } {#1} \c_false_bool }

```

(End definition for `\raiseto` and `\tothe`. These functions are documented on page 112.)

6.5 Main formatting routine

Unit input can take two forms, “literal” units (material to be typeset directly) or “symbolic” units (macro-based). Before any parsing or typesetting is carried out, a small amount of pre-parsing has to be carried out to decide which of these cases applies.

`\l_siunitx_unit_font_tl` Options which apply to the main formatting routine, and so are not tied to either symbolic or literal input.

```

121 \keys_define:nn { siunitx }
122 {
123   font-command .tl_set:N =
124     \l_siunitx_unit_font_tl ,
125   inter-unit-product .tl_set:N =
126     \l_siunitx_unit_product_tl
127 }

```

(End definition for `\l_siunitx_unit_font_tl` and `\l_siunitx_unit_product_tl`. This variable is documented on page 108.)

`\l_siunitx_unit_formatted_tl` A token list for the final formatted result: may or may not be generated by the parser, depending on the nature of the input.

```

128 \tl_new:N \l_siunitx_unit_formatted_tl

```

(End definition for `\l_siunitx_unit_formatted_tl`.)

`\siunitx_unit_format:nN` Formatting parsed units can take place either with the prefixes printed or separated out into a power of ten. This variation is handled using two separate functions: as this submodule does not really deal with numbers, formatting the numeral part here would be tricky and it is better therefore to have a mechanism to return a simple numerical power. At the same time, most uses will not want this more complex return format and so a version of the code which does not do this is also provided.

The main unit formatting routine groups all of the parsing/formatting, so that the only value altered will be the return token list. As definitions for the various unit macros are not globally created, the first step is to map over the list of names and active the unit definitions: these do different things depending on the switches set. There is then a decision to be made: is the unit input one that can be parsed (“symbolic”), or is it one containing one or more literals. In the latter case, there is still the need to convert the input into an expanded token list as some parts of the input could still be using unit macros.

Notice that for `\siunitx_unit_format:nN` a second return value from the auxiliary has to be allowed for, but is simply discarded.

```

129 \cs_new_protected:Npn \siunitx_unit_format:nN #1#2
130 {
131   \bool_set_false:N \l__siunitx_unit_prefix_power_bool
132   \__siunitx_unit_format:nNN {#1} #2 \l__siunitx_unit_tmp_fp
133 }
134 \cs_new_protected:Npn \siunitx_unit_format:nNN #1#2#3
135 {
136   \bool_set_true:N \l__siunitx_unit_prefix_power_bool
137   \__siunitx_unit_format:nNN {#1} #2 #3
138 }
139 \cs_new_protected:Npn \__siunitx_unit_format:nNN #1#2#3
140 {
141   \group_begin:
142   \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
143     { \cs_set_eq:Nc ##1 { __siunitx_unit_token_to_str:N ##1 :w } }
144   \tl_clear:N \l__siunitx_unit_formatted_tl
145   \fp_zero:N \l__siunitx_unit_prefix_fp
146   \bool_if:NTF \l__siunitx_unit_parse_bool
147     {
148       \__siunitx_unit_if_symbolic:nTF {#1}
149       {
150         \__siunitx_unit_parse:n {#1}
151         \prop_if_empty:NF \l__siunitx_unit_parsed_prop
152         { \__siunitx_unit_format_parsed: }
153       }
154       {
155         \bool_if:NTF \l__siunitx_unit_forbid_literal_bool
156         { \msg_error:nnn { siunitx } { unit / literal } {#1} }
157         { \__siunitx_unit_format_literal:n {#1} }
158       }
159     }
160   { \__siunitx_unit_format_literal:n {#1} }
161   \cs_set_protected:Npx \__siunitx_unit_format_aux:
162   {
163     \tl_set:Nn \exp_not:N #2
164     { \exp_not:V \l__siunitx_unit_formatted_tl }
165     \fp_set:Nn \exp_not:N #3
166     { \fp_use:N \l__siunitx_unit_prefix_fp }
167   }
168   \exp_after:wN \group_end:
169   \__siunitx_unit_format_aux:
170 }
171 \cs_new_protected:Npn \__siunitx_unit_format_aux: { }

```

(End definition for `\siunitx_unit_format:nN` and others. These functions are documented on page 106.)

6.6 Formatting literal units

While in literal mode no parsing occurs, there is a need to provide a few auxiliary functions to handle one or two special cases.

`__siunitx_unit_literal_power:nn` For printing literal units which are given before the unit they apply to, there is a slight rearrangement. This is ex[EXP]pandable to cover the case of creation of a PDF string.

```
172 \cs_new:Npn \__siunitx_unit_literal_power:nn #1#2 { #2 ^ {#1} }
```

(End definition for __siunitx_unit_literal_power:nn.)

__siunitx_unit_literal_special:nN When dealing with the special cases, there is an argument to absorb. This should be braced to be passed up to the user level, which is dealt with here.

```
173 \cs_new:Npn \__siunitx_unit_literal_special:nN #1#2 { #1 {#2} }
```

(End definition for __siunitx_unit_literal_special:nN.)

_siunitx_unit_format_literal:n To format literal units, there are two tasks to do. The input is x-type expanded to force any symbolic units to be converted into their literal representation: this requires setting the appropriate switch. In the resulting token list, all . and ~ tokens are then replaced by the current unit product token list. To enable this to happen correctly with a normal (active) ~, a small amount of “protection” is needed first. To cover active sub- and superscript tokens, appropriate definitions are provided at this stage. Those have to be expandable macros rather than implicit character tokens.

As with other code dealing with user input, \protected@edef is used here rather than \tl_set:Nx as L^AT_EX 2_ε robust commands may be present.

```
174 \group_begin:
175   \char_set_catcode_active:n { '~ }
176   \cs_new_protected:Npx \__siunitx_unit_format_literal:n #1
177   {
178     \group_begin:
179     \exp_not:n { \bool_set_false:N \l__siunitx_unit_parsing_bool }
180     \tl_set:Nn \exp_not:N \l__siunitx_unit_tmp_tl {#1}
181     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
182     { \token_to_str:N ^ } { ^ }
183     \tl_replace_all:Nnn \exp_not:N \l__siunitx_unit_tmp_tl
184     { \token_to_str:N _ } { \c__siunitx_unit_math_subscript_tl }
185     \char_set_active_eq:NN ^
186     \exp_not:N \__siunitx_unit_format_literal_superscript:
187     \char_set_active_eq:NN _
188     \exp_not:N \__siunitx_unit_format_literal_subscript:
189     \char_set_active_eq:NN \exp_not:N ~
190     \exp_not:N \__siunitx_unit_format_literal_tilde:
191     \exp_not:n
192     {
193       \protected@edef \l__siunitx_unit_tmp_tl
194       { \l__siunitx_unit_tmp_tl }
195       \tl_clear:N \l__siunitx_unit_formatted_tl
196       \tl_if_empty:NF \l__siunitx_unit_tmp_tl
197       {
198         \exp_after:wN \__siunitx_unit_format_literal_auxi:w
199         \l__siunitx_unit_tmp_tl .
200         \q_recursion_tail . \q_recursion_stop
201       }
202       \exp_args:NNNV \group_end:
203       \tl_set:Nn \l__siunitx_unit_formatted_tl
204       \l__siunitx_unit_formatted_tl
205     }
206   }
207 \group_end:
208 \cs_new:Npx \__siunitx_unit_format_literal_subscript: { \c__siunitx_unit_math_subscript_tl }
```

```

209 \cs_new:Npn \__siunitx_unit_format_literal_superscript: { ^ }
210 \cs_new:Npn \__siunitx_unit_format_literal_tilde: { . }

```

To introduce the font changing commands while still allowing for line breaks in literal units, a loop is needed to replace one . at a time. To also allow for division, a second loop is used within that to handle /: as a result, the separator between parts has to be tracked.

```

211 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxi:w #1 .
212 {
213   \quark_if_recursion_tail_stop:n {#1}
214   \__siunitx_unit_format_literal_auxii:n {#1}
215   \tl_set_eq:NN \l__siunitx_unit_separator_tl \l__siunitx_unit_product_tl
216   \__siunitx_unit_format_literal_auxi:w
217 }
218 \cs_set_protected:Npn \__siunitx_unit_format_literal_auxii:n #1
219 {
220   \__siunitx_unit_format_literal_auxiii:w
221   #1 / \q_recursion_tail / \q_recursion_stop
222 }
223 \cs_new_protected:Npn \__siunitx_unit_format_literal_auxiii:w #1 /
224 {
225   \quark_if_recursion_tail_stop:n {#1}
226   \__siunitx_unit_format_literal_auxiv:w #1 ^ ^ \q_stop
227   \tl_set:Nn \l__siunitx_unit_separator_tl { / }
228   \__siunitx_unit_format_literal_auxiii:w
229 }

```

Within each unit any sub- and superscript parts need to be separated out: wrapping everything in the font command is incorrect. The superscript part is relatively easy as there is no catcode issue or font command to add, while the subscript part needs a bit more work. As the user might have the two parts in either order, picking up the subscript needs to look in two places. We assume that only one is given: odd input here is simply accepted.

```

230 \use:x
231 {
232   \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxiv:w
233     ##1 ^ ##2 ^ ##3 \exp_not:N \q_stop
234   {
235     \exp_not:N \__siunitx_unit_format_literal_auxv:w
236     ##1
237     \c__siunitx_unit_math_subscript_tl
238     \c__siunitx_unit_math_subscript_tl
239     \exp_not:N \q_mark
240     ##2
241     \c__siunitx_unit_math_subscript_tl
242     \c__siunitx_unit_math_subscript_tl
243     \exp_not:N \q_stop
244   }
245   \cs_new_protected:Npn \exp_not:N \__siunitx_unit_format_literal_auxv:w
246     ##1 \c__siunitx_unit_math_subscript_tl
247     ##2 \c__siunitx_unit_math_subscript_tl ##3
248     \exp_not:N \q_mark
249     ##4 \c__siunitx_unit_math_subscript_tl
250     ##5 \c__siunitx_unit_math_subscript_tl ##6

```

```

251 \exp_not:N \q_stop
252 {
253   \tl_set:Nx \exp_not:N \l__siunitx_unit_formatted_tl
254   {
255     \exp_not:N \exp_not:V
256     \exp_not:N \l__siunitx_unit_formatted_tl
257     \exp_not:N \tl_if_empty:NF
258     \exp_not:N \l__siunitx_unit_formatted_tl
259     {
260       \exp_not:N \exp_not:V
261       \exp_not:N \l__siunitx_unit_separator_tl
262     }
263     \exp_not:N \tl_if_blank:nF {##1}
264     {
265       \exp_not:N \exp_not:V
266       \exp_not:N \l__siunitx_unit_font_tl
267       { \exp_not:N \exp_not:n {##1} }
268     }
269     \exp_not:N \tl_if_blank:nF {##4}
270     { ~ { \exp_not:N \exp_not:n {##4} } }
271     \exp_not:N \tl_if_blank:nF {##2##5}
272     {
273       \c__siunitx_unit_math_subscript_tl
274       {
275         \exp_not:N \exp_not:V
276         \exp_not:N \l__siunitx_unit_font_tl
277         { \exp_not:N \exp_not:n {##2##5} }
278       }
279     }
280   }
281 }
282 }
283 \tl_new:N \l__siunitx_unit_separator_tl

```

(End definition for `__siunitx_unit_format_literal:n` and others.)

6.7 (PDF) String creation

`\siunitx_unit_pdfstring_context:` A simple function that sets up to make units equal to their text representation.

```

284 \cs_new_protected:Npn \siunitx_unit_pdfstring_context:
285 {
286   \bool_set_false:N \l__siunitx_unit_parsing_bool
287   \seq_map_inline:Nn \l__siunitx_unit_symbolic_seq
288   { \cs_set_eq:Nc ##1 { __siunitx_unit_ \token_to_str:N ##1 :w } }
289 }

```

(End definition for `\siunitx_unit_pdfstring_context:.` This function is documented on page 108.)

6.8 Parsing symbolic units

Parsing units takes place by storing information about each unit in a `prop`. As well as the unit itself, there are various other optional data points, for example a prefix or a power. Some of these can come before the unit, others only after. The parser therefore tracks the number of units read and uses the current position to allocate data to individual units.

The result of parsing is a property list (`\l__siunitx_unit_parsed_prop`) which contains one or more entries for each unit:

- **prefix-*n*** The symbol for the prefix which applies to this unit, *e.g.* for `\kilo` with (almost certainly) would be `k`.
- **unit-*n*** The symbol for the unit itself, *e.g.* for `\metre` with (almost certainly) would be `m`.
- **power-*n*** The power which a unit is raised to. During initial parsing this will (almost certainly) be positive, but is combined with **per-*n*** to give a “fully qualified” power before any formatting takes place
- **per-*n*** Indicates that **per** applies to the current unit: stored during initial parsing then combined with **power-*n*** (and removed from the list) before further work.
- **qualifier-*n*** Any qualifier which applies to the current unit.
- **special-*n*** Any “special effect” to apply to the current unit.

`\l__siunitx_unit_sticky_per_bool` There is one option when *parsing* the input (as opposed to *formatting* for output): how to deal with `\per`.

```

290 \keys_define:nn { siunitx }
291   {
292     sticky-per .bool_set:N = \l__siunitx_unit_sticky_per_bool
293   }

```

(End definition for `\l__siunitx_unit_sticky_per_bool`.)

`\l__siunitx_unit_parsed_prop` Parsing units requires a small number of variables are available: a **prop** for the parsed units themselves, a **bool** to indicate if `\per` is active and an **int** to track how many units have be parsed.

```

294 \prop_new:N \l__siunitx_unit_parsed_prop
295 \bool_new:N \l__siunitx_unit_per_bool
296 \int_new:N \l__siunitx_unit_position_int

```

(End definition for `\l__siunitx_unit_parsed_prop`, `\l__siunitx_unit_per_bool`, and `\l__siunitx_unit_position_int`.)

`__siunitx_unit_parse:n` The main parsing function is quite simple. After initialising the variables, each symbolic unit is set up. The input is then simply inserted into the input stream: the symbolic units themselves then do the real work of placing data into the parsing system. There is then a bit of tidying up to ensure that later stages can rely on the nature of the data here.

```

297 \cs_new_protected:Npn \__siunitx_unit_parse:n #1
298   {
299     \prop_clear:N \l__siunitx_unit_parsed_prop
300     \bool_set_true:N \l__siunitx_unit_parsing_bool
301     \bool_set_false:N \l__siunitx_unit_per_bool
302     \bool_set_false:N \l__siunitx_unit_test_bool
303     \int_zero:N \l__siunitx_unit_position_int
304     \siunitx_unit_options_apply:n {#1}
305     #1
306     \int_step_inline:nn \l__siunitx_unit_position_int
307       { \__siunitx_unit_parse_finalise:n {##1} }

```



```

308     \__siunitx_unit_parse_finalise:
309 }

```

(End definition for __siunitx_unit_parse:n.)

__siunitx_unit_parse_add:nnnn In all cases, storing a data item requires setting a temporary `tl` which will be used as the key, then using this to store the value. The `tl` is set using `x`-type expansion as this will expand the unit index and any additional calculations made for this.

```

310 \cs_new_protected:Npn \__siunitx_unit_parse_add:nnnn #1#2#3#4
311 {
312     \tl_set:Nx \l__siunitx_unit_tmp_tl { #1 - #2 }
313     \prop_if_in:NVTF \l__siunitx_unit_parsed_prop
314         \l__siunitx_unit_tmp_tl
315     {
316         \msg_error:nnxx { siunitx } { unit / duplicate-part }
317         { \exp_not:n {#1} } { \token_to_str:N #3 }
318     }
319     {
320         \prop_put:NVn \l__siunitx_unit_parsed_prop
321             \l__siunitx_unit_tmp_tl {#4}
322     }
323 }

```

(End definition for __siunitx_unit_parse_add:nnnn.)

__siunitx_unit_parse_prefix:Nn Storage of the various optional items follows broadly the same pattern in each case. The data to be stored is passed along with an appropriate key name to the underlying storage system. The details for each type of item should be relatively clear. For example, prefixes have to come before their “parent” unit and so there is some adjustment to do to add them to the correct unit.

```

324 \cs_new_protected:Npn \__siunitx_unit_parse_prefix:Nn #1#2
325 {
326     \int_set:Nn \l__siunitx_unit_tmp_int { \l__siunitx_unit_position_int + 1 }
327     \__siunitx_unit_parse_add:nnnn { prefix }
328     { \int_use:N \l__siunitx_unit_tmp_int } {#1} {#2}
329 }
330 \cs_new_protected:Npn \__siunitx_unit_parse_power:nnN #1#2#3
331 {
332     \tl_set:Nx \l__siunitx_unit_tmp_tl
333         { unit- \int_use:N \l__siunitx_unit_position_int }
334     \bool_lazy_or:nnTF
335         {#3}
336         {
337             \prop_if_in_p:NV
338                 \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
339             }
340         {
341             \__siunitx_unit_parse_add:nnnn { power }
342             {
343                 \int_eval:n
344                     { \l__siunitx_unit_position_int \bool_if:NT #3 { + 1 } }
345             }
346             {#1} {#2}
347         }
348     }

```

```

348     {
349         \msg_error:nnxx { siunitx }
350         { unit / part-before-unit } { power } { \token_to_str:N #1 }
351     }
352 }
353 \cs_new_protected:Npn \__siunitx_unit_parse_qualifier:nn #1#2
354 {
355     \tl_set:Nx \l__siunitx_unit_tmp_tl
356     { unit- \int_use:N \l__siunitx_unit_position_int }
357     \prop_if_in:NVTF \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
358     {
359         \__siunitx_unit_parse_add:nnnn { qualifier }
360         { \int_use:N \l__siunitx_unit_position_int } {#1} {#2}
361     }
362     {
363         \msg_error:nnnn { siunitx }
364         { unit / part-before-unit } { qualifier } { \token_to_str:N #1 }
365     }
366 }

```

Special (exceptional) items should always come before the relevant units.

```

367 \cs_new_protected:Npn \__siunitx_unit_parse_special:n #1
368 {
369     \__siunitx_unit_parse_add:nnnn { special }
370     { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
371     {#1} {#1}
372 }

```

(End definition for `__siunitx_unit_parse_prefix:Nn` and others.)

`__siunitx_unit_parse_unit:Nn`

Parsing units is slightly more involved than the other cases: this is the one place where the tracking value is incremented. If the switch `\l__siunitx_unit_per_bool` is set true then the current unit is also reciprocal: this can only happen if `\l__siunitx_unit_sticky_per_bool` is also true, so only one test is required.

```

373 \cs_new_protected:Npn \__siunitx_unit_parse_unit:Nn #1#2
374 {
375     \int_incr:N \l__siunitx_unit_position_int
376     \__siunitx_unit_parse_add:nnnn { unit }
377     { \int_use:N \l__siunitx_unit_position_int }
378     {#1} {#2}
379     \bool_if:NT \l__siunitx_unit_per_bool
380     {
381         \__siunitx_unit_parse_add:nnnn { per }
382         { \int_use:N \l__siunitx_unit_position_int }
383         { \per } { true }
384     }
385 }

```

(End definition for `__siunitx_unit_parse_unit:Nn`.)

`__siunitx_unit_parse_per:`

Storing the `\per` command requires adding a data item separate from the power which applies: this makes later formatting much more straight-forward. This data could in principle be combined with the `power`, but depending on the output format required that may make life more complex. Thus this information is stored separately for later

retrieval. If `\per` is set to be “sticky” then after parsing the first occurrence, any further uses are in error.

```

386 \cs_new_protected:Npn \__siunitx_unit_parse_per:
387 {
388   \bool_if:NTF \l__siunitx_unit_sticky_per_bool
389   {
390     \bool_set_true:N \l__siunitx_unit_per_bool
391     \cs_set_protected:Npn \per
392     { \msg_error:nn { siunitx } { unit / duplicate-sticky-per } }
393   }
394   {
395     \__siunitx_unit_parse_add:nnnn
396     { per } { \int_eval:n { \l__siunitx_unit_position_int + 1 } }
397     { \per } { true }
398   }
399 }

```

(End definition for `__siunitx_unit_parse_per:.`)

`__siunitx_unit_parse_finalise:n` If `\per` applies to the current unit, the power needs to be multiplied by -1 . That is done using an `fp` operation so that non-integer powers are supported. The flag for `\per` is also removed as this means we don’t have to check that the original power was positive. To be on the safe side, there is a check for a trivial power at this stage.

```

400 \cs_new_protected:Npn \__siunitx_unit_parse_finalise:n #1
401 {
402   \tl_set:Nx \l__siunitx_unit_tmp_tl { per- #1 }
403   \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
404   {
405     \prop_remove:NV \l__siunitx_unit_parsed_prop
406     \l__siunitx_unit_tmp_tl
407     \tl_set:Nx \l__siunitx_unit_tmp_tl { power- #1 }
408     \prop_get:NVNTF
409     \l__siunitx_unit_parsed_prop
410     \l__siunitx_unit_tmp_tl
411     \l__siunitx_unit_part_tl
412     {
413       \tl_set:Nx \l__siunitx_unit_part_tl
414       { \fp_eval:n { \l__siunitx_unit_part_tl * -1 } }
415       \fp_compare:nNnTF \l__siunitx_unit_part_tl = 1
416       {
417         \prop_remove:NV \l__siunitx_unit_parsed_prop
418         \l__siunitx_unit_tmp_tl
419       }
420       {
421         \prop_put:NVV \l__siunitx_unit_parsed_prop
422         \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
423       }
424     }
425     {
426       \prop_put:NVN \l__siunitx_unit_parsed_prop
427       \l__siunitx_unit_tmp_tl { -1 }
428     }
429   }
430 }

```

(End definition for _siunitx_unit_parse_finalise:n.)

_siunitx_unit_parse_finalise: The final task is to check that there is not a “dangling” power or prefix: these are added to the “next” unit so are easy to test for.

```

431 \cs_new_protected:Npn \_siunitx_unit_parse_finalise:
432 {
433   \clist_map_inline:nn { per , power , prefix }
434   {
435     \tl_set:Nx \l__siunitx_unit_tmp_tl
436     { ##1 - \int_eval:n { \l__siunitx_unit_position_int + 1 } }
437     \prop_if_in:NVT \l__siunitx_unit_parsed_prop \l__siunitx_unit_tmp_tl
438     { \msg_error:nnn { siunitx } { unit / dangling-part } { ##1 } }
439   }
440 }

```

(End definition for _siunitx_unit_parse_finalise:.)

6.9 Formatting parsed units

Set up the options which apply to formatting.

```

\_siunitx_unit_denominator_bracket_bool
\_siunitx_unit_forbid_literal_bool
\_siunitx_unit_fraction_function_tl
\_siunitx_unit_bracket_close_tl
\_siunitx_unit_bracket_open_tl
\_siunitx_unit_parse_bool
\_siunitx_unit_per_symbol_tl
\_siunitx_unit_qualifier_mode_tl
\_siunitx_unit_qualifier_phrase_tl
441 \keys_define:nn { siunitx }
442 {
443   bracket-unit-denominator .bool_set:N =
444   \l__siunitx_unit_denominator_bracket_bool ,
445   forbid-literal-units .bool_set:N =
446   \l__siunitx_unit_forbid_literal_bool ,
447   fraction-command .tl_set:N =
448   \l__siunitx_unit_fraction_function_tl ,
449   parse-units .bool_set:N =
450   \l__siunitx_unit_parse_bool ,
451   per-mode .choice: ,
452   per-mode / fraction .code:n =
453   {
454     \bool_set_false:N \l__siunitx_unit_autofrac_bool
455     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
456     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
457     \bool_set_true:N \l__siunitx_unit_two_part_bool
458   } ,
459   per-mode / power .code:n =
460   {
461     \bool_set_false:N \l__siunitx_unit_autofrac_bool
462     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
463     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
464     \bool_set_false:N \l__siunitx_unit_two_part_bool
465   } ,
466   per-mode / power-positive-first .code:n =
467   {
468     \bool_set_false:N \l__siunitx_unit_autofrac_bool
469     \bool_set_false:N \l__siunitx_unit_per_symbol_bool
470     \bool_set_false:N \l__siunitx_unit_powers_positive_bool
471     \bool_set_true:N \l__siunitx_unit_two_part_bool
472   } ,
473   per-mode / repeated-symbol .code:n =
474   {

```

```

475     \bool_set_false:N \l__siunitx_unit_autofrac_bool
476     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
477     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
478     \bool_set_false:N \l__siunitx_unit_two_part_bool
479   } ,
480   per-mode / symbol .code:n =
481   {
482     \bool_set_false:N \l__siunitx_unit_autofrac_bool
483     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
484     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
485     \bool_set_true:N \l__siunitx_unit_two_part_bool
486   } ,
487   per-mode / symbol-or-fraction .code:n =
488   {
489     \bool_set_true:N \l__siunitx_unit_autofrac_bool
490     \bool_set_true:N \l__siunitx_unit_per_symbol_bool
491     \bool_set_true:N \l__siunitx_unit_powers_positive_bool
492     \bool_set_true:N \l__siunitx_unit_two_part_bool
493   } ,
494   per-symbol .tl_set:N =
495     \l__siunitx_unit_per_symbol_tl ,
496   qualifier-mode .choices:nn =
497     { bracket , combine , phrase , subscript }
498     { \tl_set_eq:NN \l__siunitx_unit_qualifier_mode_tl \l_keys_choice_tl } ,
499   qualifier-phrase .tl_set:N =
500     \l__siunitx_unit_qualifier_phrase_tl ,
501   unit-close-bracket .tl_set:N =
502     \l__siunitx_unit_bracket_close_tl ,
503   unit-open-bracket .tl_set:N =
504     \l__siunitx_unit_bracket_open_tl
505 }

```

(End definition for `\l__siunitx_unit_denominator_bracket_bool` and others.)

`\l__siunitx_unit_bracket_bool` A flag to indicate that the unit currently under construction will require brackets if a power is added.

```
506 \bool_new:N \l__siunitx_unit_bracket_bool
```

(End definition for `\l__siunitx_unit_bracket_bool`.)

`\l__siunitx_unit_font_bool` A flag to control when font wrapping is applied to the output.

```
507 \bool_new:N \l__siunitx_unit_font_bool
```

(End definition for `\l__siunitx_unit_font_bool`.)

`\l__siunitx_unit_autofrac_bool`
`\l__siunitx_unit_powers_positive_bool`
`\l__siunitx_unit_per_symbol_bool`
`\l__siunitx_unit_two_part_bool` Dealing with the various ways that reciprocal (`\per`) can be handled requires a few different switches.

```

508 \bool_new:N \l__siunitx_unit_autofrac_bool
509 \bool_new:N \l__siunitx_unit_per_symbol_bool
510 \bool_new:N \l__siunitx_unit_powers_positive_bool
511 \bool_new:N \l__siunitx_unit_two_part_bool

```

(End definition for `\l__siunitx_unit_autofrac_bool` and others.)

`\l_siunitx_unit_numerator_bool` Indicates that the current unit should go into the numerator when splitting into two parts (fractions or other “sorted” styles).

```

512 \bool_new:N \l__siunitx_unit_numerator_bool
(End definition for \l_siunitx_unit_numerator_bool.)

```

`\l_siunitx_unit_qualifier_mode_tl` For storing the text of options which are best handled by picking function names.

```

513 \tl_new:N \l__siunitx_unit_qualifier_mode_tl
(End definition for \l__siunitx_unit_qualifier_mode_tl.)

```

`\l_siunitx_unit_prefix_power_bool` Used to determine if prefixes are converted into powers. Note that while this may be set as an option “higher up”, at this point it is handled as an internal switch (see the two formatting interfaces for reasons).

```

514 \bool_new:N \l__siunitx_unit_prefix_power_bool
(End definition for \l_siunitx_unit_prefix_power_bool.)

```

`\l__siunitx_unit_prefix_fp` When converting prefixes to powers, the calculations are done as an fp.

```

515 \fp_new:N \l__siunitx_unit_prefix_fp
(End definition for \l__siunitx_unit_prefix_fp.)

```

`\l__siunitx_unit_current_tl`
`\l__siunitx_unit_part_tl` Building up the (partial) formatted unit requires some token list storage. Each part of the unit combination that is recovered also has to be placed in a token list: this is a dedicated one to leave the scratch variables available.

```

516 \tl_new:N \l__siunitx_unit_current_tl
517 \tl_new:N \l__siunitx_unit_part_tl
(End definition for \l__siunitx_unit_current_tl and \l__siunitx_unit_part_tl.)

```

`\l_siunitx_unit_denominator_tl` For fraction-like units, space is needed for the denominator as well as the numerator (which is handled using `\l__siunitx_unit_formatted_tl`).

```

518 \tl_new:N \l__siunitx_unit_denominator_tl
(End definition for \l__siunitx_unit_denominator_tl.)

```

`\l__siunitx_unit_total_int` The formatting routine needs to know both the total number of units and the current unit. Thus an int is required in addition to `\l__siunitx_unit_position_int`.

```

519 \int_new:N \l__siunitx_unit_total_int
(End definition for \l__siunitx_unit_total_int.)

```

`_siunitx_unit_format_parsed:`
`_siunitx_unit_format_parsed_aux:n` The main formatting routine is essentially a loop over each position, reading the various parts of the unit to build up complete unit combination.

```

520 \cs_new_protected:Npn \_siunitx_unit_format_parsed:
521 {
522   \int_set_eq:NN \l__siunitx_unit_total_int \l__siunitx_unit_position_int
523   \tl_clear:N \l__siunitx_unit_denominator_tl
524   \tl_clear:N \l__siunitx_unit_formatted_tl
525   \fp_zero:N \l__siunitx_unit_prefix_fp
526   \int_zero:N \l__siunitx_unit_position_int
527   \int_do_while:nNnn
528     \l__siunitx_unit_position_int < \l__siunitx_unit_total_int
529     {

```

```

530     \bool_set_false:N \l__siunitx_unit_bracket_bool
531     \tl_clear:N \l__siunitx_unit_current_tl
532     \bool_set_false:N \l__siunitx_unit_font_bool
533     \bool_set_true:N \l__siunitx_unit_numerator_bool
534     \int_incr:N \l__siunitx_unit_position_int
535     \clist_map_inline:nn { prefix , unit , qualifier , power , special }
536       { \__siunitx_unit_format_parsed_aux:n {##1} }
537     \__siunitx_unit_format_output:
538   }
539   \__siunitx_unit_format_finalise:
540 }
541 \cs_new_protected:Npn \__siunitx_unit_format_parsed_aux:n #1
542 {
543   \tl_set:Nx \l__siunitx_unit_tmp_tl
544     { #1 - \int_use:N \l__siunitx_unit_position_int }
545   \prop_get:NVNT \l__siunitx_unit_parsed_prop
546     \l__siunitx_unit_tmp_tl \l__siunitx_unit_part_tl
547   { \use:c { __siunitx_unit_format_ #1 : } }
548 }

```

(End definition for `__siunitx_unit_format_parsed:` and `__siunitx_unit_format_parsed_aux:n`.)

`__siunitx_unit_format_bracket:N`

A quick utility function which wraps up a token list variable in brackets if they are required.

```

549 \cs_new:Npn \__siunitx_unit_format_bracket:N #1
550 {
551   \bool_if:NTF \l__siunitx_unit_bracket_bool
552   {
553     \exp_not:V \l__siunitx_unit_bracket_open_tl
554     \exp_not:V #1
555     \exp_not:V \l__siunitx_unit_bracket_close_tl
556   }
557   { \exp_not:V #1 }
558 }

```

(End definition for `__siunitx_unit_format_bracket:N`.)

`__siunitx_unit_format_power:`

`__siunitx_unit_format_power_aux:wTF`

`__siunitx_unit_format_power_positive:`

`__siunitx_unit_format_power_negative:`

`__siunitx_unit_format_power_negative_aux:w`

`__siunitx_unit_format_power_superscript:`

Formatting powers requires a test for negative numbers and depending on output format requests some adjustment to the stored value. This could be done using an `fp` function, but that would be slow compared to a dedicated if lower-level approach based on delimited arguments.

```

559 \cs_new_protected:Npn \__siunitx_unit_format_power:
560 {
561   \__siunitx_unit_format_font:
562   \exp_after:wN \__siunitx_unit_format_power_aux:wTF
563     \l__siunitx_unit_part_tl - \q_stop
564     { \__siunitx_unit_format_power_negative: }
565     { \__siunitx_unit_format_power_positive: }
566 }
567 \cs_new:Npn \__siunitx_unit_format_power_aux:wTF #1 - #2 \q_stop
568 { \tl_if_empty:nTF {#1} }

```

In the case of positive powers, there is little to do: add the power as a subscript (must be required as the parser ensures it's $\neq 1$).

```

569 \cs_new_protected:Npn \__siunitx_unit_format_power_positive:
570 { \__siunitx_unit_format_power_superscript: }

```

Dealing with negative powers starts by flipping the switch used to track where in the final output the current part should get added to. For the case where the output is fraction-like, strip off the \sim then ensure that the result is not the trivial power 1. Assuming all is well, addition to the current unit combination goes ahead.

```

571 \cs_new_protected:Npn \__siunitx_unit_format_power_negative:
572 {
573   \bool_set_false:N \l__siunitx_unit_numerator_bool
574   \bool_if:NTF \l__siunitx_unit_powers_positive_bool
575   {
576     \tl_set:Nx \l__siunitx_unit_part_tl
577     {
578       \exp_after:wN \__siunitx_unit_format_power_negative_aux:w
579       \l__siunitx_unit_part_tl \q_stop
580     }
581     \str_if_eq:VnF \l__siunitx_unit_part_tl { 1 }
582     { \__siunitx_unit_format_power_superscript: }
583   }
584   { \__siunitx_unit_format_power_superscript: }
585 }
586 \cs_new:Npn \__siunitx_unit_format_power_negative_aux:w - #1 \q_stop
587 { \exp_not:n {#1} }

```

Adding the power as a superscript has the slight complication that there is the possibility of needing some brackets. The superscript itself uses `\sp` as that avoids any category code issues and also allows redirection at a higher level more readily.

```

588 \cs_new_protected:Npn \__siunitx_unit_format_power_superscript:
589 {
590   \tl_set:Nx \l__siunitx_unit_current_tl
591   {
592     \__siunitx_unit_format_bracket:N \l__siunitx_unit_current_tl
593     ~ { \exp_not:V \l__siunitx_unit_part_tl }
594   }
595   \bool_set_false:N \l__siunitx_unit_bracket_bool
596 }

```

(End definition for `__siunitx_unit_format_power:` and others.)

`__siunitx_unit_format_prefix:` Formatting for prefixes depends on whether they are to be expressed as symbols or collected up to be returned as a power of 10. The latter case requires a bit of processing, which includes checking that the conversion is possible and allowing for any power that applies to the current unit.

```

597 \cs_new_protected:Npn \__siunitx_unit_format_prefix:
598 {
599   \bool_if:NTF \l__siunitx_unit_prefix_power_bool
600   { \__siunitx_unit_format_prefix_power: }
601   { \__siunitx_unit_format_prefix_symbol: }
602 }
603 \cs_new_protected:Npn \__siunitx_unit_format_prefix_power:
604 {
605   \prop_get:NVNTF \l__siunitx_unit_prefixes_forward_prop
606   \l__siunitx_unit_part_tl \l__siunitx_unit_part_tl

```



```

607 {
608   \tl_set:Nx \l__siunitx_unit_tmp_tl
609   { power- \int_use:N \l__siunitx_unit_position_int }
610   \prop_get:NVNF \l__siunitx_unit_parsed_prop
611   \l__siunitx_unit_tmp_tl \l__siunitx_unit_tmp_tl
612   { \tl_set:Nn \l__siunitx_unit_tmp_tl { 1 } }
613   \fp_add:Nn \l__siunitx_unit_prefix_fp
614   { \l__siunitx_unit_tmp_tl * \l__siunitx_unit_part_tl }
615 }
616 { \__siunitx_unit_format_prefix_symbol: }
617 }
618 \cs_new_protected:Npn \__siunitx_unit_format_prefix_symbol:
619 { \tl_set_eq:NN \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl }

(End definition for \__siunitx_unit_format_prefix:, \__siunitx_unit_format_prefix_power:, and
\__siunitx_unit_format_prefix_symbol:.)

```

`__siunitx_unit_format_qualifier:` There are various ways that a qualifier can be added to the output. The idea here is to modify the “base” text appropriately and then add to the current unit. Notice that when the qualifier is just treated as “text”, the auxiliary is actually a no-op.

```

620 \cs_new_protected:Npn \__siunitx_unit_format_qualifier:
621 {
622   \use:c
623   {
624     __siunitx_unit_format_qualifier_
625     \l__siunitx_unit_qualifier_mode_tl :
626   }
627   \tl_put_right:NV \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
628 }
629 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_bracket:
630 {
631   \__siunitx_unit_format_font:
632   \tl_set:Nx \l__siunitx_unit_part_tl
633   {
634     \exp_not:V \l__siunitx_unit_bracket_open_tl
635     \exp_not:V \l__siunitx_unit_font_tl
636     { \exp_not:V \l__siunitx_unit_part_tl }
637     \exp_not:V \l__siunitx_unit_bracket_close_tl
638   }
639 }
640 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_combine: { }
641 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_phrase:
642 {
643   \__siunitx_unit_format_font:
644   \tl_set:Nx \l__siunitx_unit_part_tl
645   {
646     \exp_not:V \l__siunitx_unit_qualifier_phrase_tl
647     \exp_not:V \l__siunitx_unit_font_tl
648     { \exp_not:V \l__siunitx_unit_part_tl }
649   }
650 }
651 \cs_new_protected:Npn \__siunitx_unit_format_qualifier_subscript:
652 {
653   \__siunitx_unit_format_font:

```

```

654 \tl_set:Nx \l__siunitx_unit_part_tl
655 {
656   \c__siunitx_unit_math_subscript_tl
657   {
658     \exp_not:V \l_siunitx_unit_font_tl
659     { \exp_not:V \l__siunitx_unit_part_tl }
660   }
661 }
662 }

```

(End definition for `__siunitx_unit_format_qualifier:` and others.)

`__siunitx_unit_format_special:` Any special odds and ends are handled by simply making the current combination into an argument for the recovered code. Font control needs to be *inside* the special formatting here.

```

663 \cs_new_protected:Npn \__siunitx_unit_format_special:
664 {
665   \tl_set:Nx \l__siunitx_unit_current_tl
666   {
667     \exp_not:V \l__siunitx_unit_part_tl
668     {
669       \bool_if:NTF \l__siunitx_unit_font_bool
670       { \use:n }
671       { \exp_not:V \l_siunitx_unit_font_tl }
672       { \exp_not:V \l__siunitx_unit_current_tl }
673     }
674   }
675   \bool_set_true:N \l__siunitx_unit_font_bool
676 }

```

(End definition for `__siunitx_unit_format_special:.`)

`__siunitx_unit_format_unit:` A very simple task: add the unit to the output currently being constructed.

```

677 \cs_new_protected:Npn \__siunitx_unit_format_unit:
678 {
679   \tl_put_right:NV
680   \l__siunitx_unit_current_tl \l__siunitx_unit_part_tl
681 }

```

(End definition for `__siunitx_unit_format_unit:.`)

`__siunitx_unit_format_output:` The first step here is to make a choice based on whether the current part should be stored as part of the numerator or denominator of a fraction. In all cases, if the switch `__siunitx_unit_format_output_aux:` is true then life is simple: add the current part to the numerator with a standard separator

```

\__siunitx_unit_format_output_aux:nn
\__siunitx_unit_format_output_aux:nV
\__siunitx_unit_format_output_aux:nv
682 \cs_new_protected:Npn \__siunitx_unit_format_output:
683 {
684   \__siunitx_unit_format_font:
685   \bool_set_false:N \l__siunitx_unit_bracket_bool
686   \use:c
687   {
688     \__siunitx_unit_format_output_
689     \bool_if:NTF \l__siunitx_unit_numerator_bool
690     { aux: }

```

```

691         { denominator: }
692     }
693 }
694 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:
695 {
696     \__siunitx_unit_format_output_aux:nV { formatted }
697     \l__siunitx_unit_product_tl
698 }

```

There are a few things to worry about at this stage if the current part is in the denominator. Powers have already been dealt with and some formatting outcomes only need a branch at the final point of building the entire unit. That means that there are three possible outcomes here: if collecting two separate parts, add to the denominator with a product separator, or if only building one token list there may be a need to use a symbol separator. When the `repeated-symbol` option is in use there may be a need to add a leading 1 to the output in the case where the first unit is in the denominator: that can be picked up by looking for empty output in combination with the flag for using a symbol in the output but not a two-part strategy.

```

699 \cs_new_protected:Npn \__siunitx_unit_format_output_denominator:
700 {
701     \bool_if:NTF \l__siunitx_unit_two_part_bool
702     {
703         \bool_lazy_and:nnT
704         { \l__siunitx_unit_denominator_bracket_bool }
705         { ! \tl_if_empty_p:N \l__siunitx_unit_denominator_tl }
706         { \bool_set_true:N \l__siunitx_unit_bracket_bool }
707         \__siunitx_unit_format_output_aux:nV { denominator }
708         \l__siunitx_unit_product_tl
709     }
710     {
711         \bool_lazy_and:nnT
712         { \l__siunitx_unit_per_symbol_bool }
713         { \tl_if_empty_p:N \l__siunitx_unit_formatted_tl }
714         { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
715         \__siunitx_unit_format_output_aux:nv { formatted }
716         {
717             \l__siunitx_unit_
718             \bool_if:NTF \l__siunitx_unit_per_symbol_bool
719             { per_symbol }
720             { product }
721             _tl
722         }
723     }
724 }
725 \cs_new_protected:Npn \__siunitx_unit_format_output_aux:nn #1#2
726 {
727     \tl_set:cx { \l__siunitx_unit_ #1 _tl }
728     {
729         \exp_not:v { \l__siunitx_unit_ #1 _tl }
730         \tl_if_empty:cF { \l__siunitx_unit_ #1 _tl }
731         { \exp_not:n {#2} }
732         \exp_not:V \l__siunitx_unit_current_tl
733     }
734 }

```

```
735 \cs_generate_variant:Nn \__siunitx_unit_format_output_aux:n { nV , nv }
```

(End definition for `__siunitx_unit_format_output:` and others.)

`__siunitx_unit_format_font:` A short auxiliary which checks if the font has been applied to the main part of the output: if not, add it and set the flag.

```
736 \cs_new_protected:Npn \__siunitx_unit_format_font:
737 {
738   \bool_if:NF \l__siunitx_unit_font_bool
739   {
740     \tl_set:Nx \l__siunitx_unit_current_tl
741     {
742       \exp_not:V \l_siunitx_unit_font_tl
743       { \exp_not:V \l__siunitx_unit_current_tl }
744     }
745     \bool_set_true:N \l__siunitx_unit_font_bool
746   }
747 }
```

(End definition for `__siunitx_unit_format_font:`)

`__siunitx_unit_format_finalise:` Finalising the unit format is really about picking up the cases involving fractions: these require assembly of the parts with the need to add additional material in some cases

```
\__siunitx_unit_format_finalise_autofrac:
\__siunitx_unit_format_finalise_fractional:
\__siunitx_unit_format_finalise_power:
748 \cs_new_protected:Npn \__siunitx_unit_format_finalise:
749 {
750   \tl_if_empty:NF \l__siunitx_unit_denominator_tl
751   {
752     \bool_if:NTF \l__siunitx_unit_powers_positive_bool
753     { \__siunitx_unit_format_finalise_fractional: }
754     { \__siunitx_unit_format_finalise_power: }
755   }
756 }
```

For fraction-like output, there are three possible choices and two actual styles. In all cases, if the numerator is empty then it is set here to 1. To deal with the “auto-format” case, the two styles (fraction and symbol) are handled in auxiliaries: this allows both to be used at the same time! Beyond that, the key here is to use a single `\tl_set:Nx` to keep down the number of assignments.

```
757 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fractional:
758 {
759   \tl_if_empty:NT \l__siunitx_unit_formatted_tl
760   { \tl_set:Nn \l__siunitx_unit_formatted_tl { 1 } }
761   \bool_if:NTF \l__siunitx_unit_autofrac_bool
762   { \__siunitx_unit_format_finalise_autofrac: }
763   {
764     \bool_if:NTF \l__siunitx_unit_per_symbol_bool
765     { \__siunitx_unit_format_finalise_symbol: }
766     { \__siunitx_unit_format_finalise_fraction: }
767   }
768 }
```

For the “auto-selected” fraction method, the two other auxiliary functions are used to do both forms of formatting. So that everything required is available, this needs one group so that the second auxiliary receives the correct input. After that it is just a case of applying `\mathchoice` to the formatted output.

```

769 \cs_new_protected:Npn \__siunitx_unit_format_finalise_autofrac:
770 {
771   \group_begin:
772   \__siunitx_unit_format_finalise_fraction:
773   \exp_args:NNNV \group_end:
774   \tl_set:Nn \l__siunitx_unit_tmp_tl \l__siunitx_unit_formatted_tl
775   \__siunitx_unit_format_finalise_symbol:
776   \tl_set:Nx \l__siunitx_unit_formatted_tl
777   {
778     \mathchoice
779     { \exp_not:V \l__siunitx_unit_tmp_tl }
780     { \exp_not:V \l__siunitx_unit_formatted_tl }
781     { \exp_not:V \l__siunitx_unit_formatted_tl }
782     { \exp_not:V \l__siunitx_unit_formatted_tl }
783   }
784 }

```

When using a fraction function the two parts are now assembled.

```

785 \cs_new_protected:Npn \__siunitx_unit_format_finalise_fraction:
786 {
787   \tl_set:Nx \l__siunitx_unit_formatted_tl
788   {
789     \exp_not:V \l__siunitx_unit_fraction_function_tl
790     { \exp_not:V \l__siunitx_unit_formatted_tl }
791     { \exp_not:V \l__siunitx_unit_denominator_tl }
792   }
793 }
794 \cs_new_protected:Npn \__siunitx_unit_format_finalise_symbol:
795 {
796   \tl_set:Nx \l__siunitx_unit_formatted_tl
797   {
798     \exp_not:V \l__siunitx_unit_formatted_tl
799     \exp_not:V \l__siunitx_unit_per_symbol_tl
800     \__siunitx_unit_format_bracket:N \l__siunitx_unit_denominator_tl
801   }
802 }

```

In the case of sorted powers, there is a test to make sure there was at least one positive power, and if so a simple join of the two parts with the appropriate product.

```

803 \cs_new_protected:Npn \__siunitx_unit_format_finalise_power:
804 {
805   \tl_if_empty:NTF \l__siunitx_unit_formatted_tl
806   {
807     \tl_set_eq:NN
808     \l__siunitx_unit_formatted_tl
809     \l__siunitx_unit_denominator_tl
810   }
811   {
812     \tl_set:Nx \l__siunitx_unit_formatted_tl
813     {
814       \exp_not:V \l__siunitx_unit_formatted_tl
815       \exp_not:V \l__siunitx_unit_product_tl
816       \exp_not:V \l__siunitx_unit_denominator_tl
817     }
818   }

```

```
819 }
```

(End definition for `_siunitx_unit_format_finalise:` and others.)

6.10 Non-Latin character support

```
\_siunitx_unit_non_latin:n
\_siunitx_unit_non_latin:nnnn
```

A small amount of code to make it convenient to include non-Latin characters in units without having to directly include them in the sources directly.

```
820 \bool_lazy_or:nnTF
821 { \sys_if_engine luatex_p: }
822 { \sys_if_engine xetex_p: }
823 {
824   \cs_new:Npn \_siunitx_unit_non_latin:n #1
825     { \char_generate:nn {#1} { \char_value_catcode:n {#1} } }
826 }
827 {
828   \cs_new:Npn \_siunitx_unit_non_latin:n #1
829     {
830       \exp_last_unbraced:Nf \_siunitx_unit_non_latin:nnnn
831       { \char_codepoint_to_bytes:n {#1} }
832     }
833   \cs_new:Npn \_siunitx_unit_non_latin:nnnn #1#2#3#4
834     {
835       \exp_after:wN \exp_after:wN \exp_after:wN
836       \exp_not:N \char_generate:nn {#1} { 13 }
837       \exp_after:wN \exp_after:wN \exp_after:wN
838       \exp_not:N \char_generate:nn {#2} { 13 }
839     }
840 }
```

(End definition for `_siunitx_unit_non_latin:n` and `_siunitx_unit_non_latin:nnnn`.)

6.11 Pre-defined unit components

Quite a number of units can be predefined: while this is a code-level module, there is little point having a unit parser which does not start off able to parse any units!

\kilogram The basic SI units: technically the correct spelling is `\metre` but US users tend to use `\meter`.

```
\metre
\meter 841 \siunitx_declare_unit:Nn \kilogram { \kilo \gram }
\mole   842 \siunitx_declare_unit:Nn \metre   { m }
\kelvin 843 \siunitx_declare_unit:Nn \meter   { \metre }
\candela 844 \siunitx_declare_unit:Nn \mole    { mol }
\second 845 \siunitx_declare_unit:Nn \second   { s }
\ampere  846 \siunitx_declare_unit:Nn \ampere   { A }
          847 \siunitx_declare_unit:Nn \kelvin   { K }
          848 \siunitx_declare_unit:Nn \candela   { cd }
```

(End definition for `\kilogram` and others. These functions are documented on page 109.)

\gram The gram is an odd unit as it is needed for the base unit kilogram.

```
849 \siunitx_declare_unit:Nn \gram { g }
```

(End definition for `\gram`. This function is documented on page 109.)

\yocto The various SI multiple prefixes are defined here: first the small ones.

```

\zepto 850 \siunitx_declare_prefix:Nnn \yocto { -24 } { y }
\atto 851 \siunitx_declare_prefix:Nnn \zepto { -21 } { z }
\femto 852 \siunitx_declare_prefix:Nnn \atto { -18 } { a }
\pico 853 \siunitx_declare_prefix:Nnn \femto { -15 } { f }
\nano 854 \siunitx_declare_prefix:Nnn \pico { -12 } { p }
\micro 855 \siunitx_declare_prefix:Nnn \nano { -9 } { n }
\milli 856 \siunitx_declare_prefix:Nnn \micro { -6 } { \_siunitx_unit_non_latin:n { "03BC } }
\centi 857 \siunitx_declare_prefix:Nnn \milli { -3 } { m }
\deci 858 \siunitx_declare_prefix:Nnn \centi { -2 } { c }
859 \siunitx_declare_prefix:Nnn \deci { -1 } { d }

```

(End definition for \yocto and others. These functions are documented on page 109.)

\deca Now the large ones.

```

\deka 860 \siunitx_declare_prefix:Nnn \deca { 1 } { da }
\hecto 861 \siunitx_declare_prefix:Nnn \deka { 1 } { da }
\kilo 862 \siunitx_declare_prefix:Nnn \hecto { 2 } { h }
\mega 863 \siunitx_declare_prefix:Nnn \kilo { 3 } { k }
\giga 864 \siunitx_declare_prefix:Nnn \mega { 6 } { M }
\tera 865 \siunitx_declare_prefix:Nnn \giga { 9 } { G }
\peta 866 \siunitx_declare_prefix:Nnn \tera { 12 } { T }
\exa 867 \siunitx_declare_prefix:Nnn \peta { 15 } { P }
868 \siunitx_declare_prefix:Nnn \exa { 18 } { E }
\zetta 869 \siunitx_declare_prefix:Nnn \zetta { 21 } { Z }
\yotta 870 \siunitx_declare_prefix:Nnn \yotta { 24 } { Y }

```

(End definition for \deca and others. These functions are documented on page 109.)

\becquerel Named derived units: first half of alphabet.

```

\degreeCelsius 871 \siunitx_declare_unit:Nn \becquerel { Bq }
\coulomb 872 \siunitx_declare_unit:Nn \degreeCelsius
\farad 873 { \ensuremath { { } ^ { \circ } } } \kern -\scriptspace C }
\gray 874 \siunitx_declare_unit:Nn \coulomb { C }
\hertz 875 \siunitx_declare_unit:Nn \farad { F }
\henry 876 \siunitx_declare_unit:Nn \gray { Gy }
\joule 877 \siunitx_declare_unit:Nn \hertz { Hz }
\katal 878 \siunitx_declare_unit:Nn \henry { H }
\lumen 879 \siunitx_declare_unit:Nn \joule { J }
\lux 880 \siunitx_declare_unit:Nn \katal { kat }
881 \siunitx_declare_unit:Nn \lumen { lm }
882 \siunitx_declare_unit:Nn \lux { lx }

```

(End definition for \becquerel and others. These functions are documented on page 110.)

\newton Named derived units: second half of alphabet.

```

\ohm 883 \siunitx_declare_unit:Nn \newton { N }
\pascal 884 \siunitx_declare_unit:Nx \ohm { \_siunitx_unit_non_latin:n { "03A9 } }
\radian 885 \siunitx_declare_unit:Nn \pascal { Pa }
\siemens 886 \siunitx_declare_unit:Nn \radian { rad }
\sievert 887 \siunitx_declare_unit:Nn \siemens { S }
\steradian 888 \siunitx_declare_unit:Nn \sievert { Sv }
\tesla 889 \siunitx_declare_unit:Nn \steradian { sr }
\volt 890 \siunitx_declare_unit:Nn \tesla { T }
\watt
\weber

```

```

891 \siunitx_declare_unit:Nn \volt      { V }
892 \siunitx_declare_unit:Nn \watt      { W }
893 \siunitx_declare_unit:Nn \weber     { Wb }

```

(End definition for `\newton` and others. These functions are documented on page 110.)

`\day` Non-SI, but accepted for general use. Once again there are two spellings, here for litre
`\hectare` and with different output in this case.

```

\hour      894 \siunitx_declare_unit:Nn \day      { d }
\litre     895 \siunitx_declare_unit:Nn \hectare { ha }
\liter     896 \siunitx_declare_unit:Nn \hour     { h }
\minute    897 \siunitx_declare_unit:Nn \litre   { L }
\tonne     898 \siunitx_declare_unit:Nn \liter    { \litre }
           899 \siunitx_declare_unit:Nn \minute { min }
           900 \siunitx_declare_unit:Nn \tonne   { t }

```

(End definition for `\day` and others. These functions are documented on page 110.)

`\arcminute` Arc units: again, non-SI, but accepted for general use.

```

\arcsecond 901 \siunitx_declare_unit:Nn \arcminute { { } ' }
\degree     902 \siunitx_declare_unit:Nn \arcsecond { { } '' }
           903 \siunitx_declare_unit:Nn \degree   { { } ^ { \circ } }

```

(End definition for `\arcminute`, `\arcsecond`, and `\degree`. These functions are documented on page 110.)

`\astronomicalunit` A few units based on physical measurements exist: these ones are accepted for use with
`\atomicmassunit` the International System.

```

\dalton     904 \siunitx_declare_unit:Nn \astronomicalunit { au }
\electronvolt 905 \siunitx_declare_unit:Nn \atomicmassunit { u }
           906 \siunitx_declare_unit:Nn \dalton      { Da }
           907 \siunitx_declare_unit:Nn \electronvolt { eV }

```

(End definition for `\astronomicalunit` and others. These functions are documented on page 111.)

`\nuaction` Natural units based on physical constants.

```

\numass     908 \siunitx_declare_unit:Nn \nuaction { \ensuremath { \mathit { \hbar } } }
\nuspeed    909 \siunitx_declare_unit:Nx \numass
\nutime     910 {
           911   \exp_not:N \ensuremath
           912   {
           913     \exp_not:N \mathit { m }
           914     \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { e } } }
           915   }
           916 }
           917 \siunitx_declare_unit:Nx \nuspeed
           918 {
           919   \exp_not:N \ensuremath
           920   { \exp_not:N \mathit { c } \c__siunitx_unit_math_subscript_tl { 0 } }
           921   }
           922 \siunitx_declare_unit:Nn \nutime
           923 { \numass \per \numass \per \nuspeed \squared }

```

(End definition for `\nuaction` and others. These functions are documented on page 111.)

\auaction Atomic units based on physical constants.

\aucharge 924 \siunitx_declare_unit:Nn \auaction { \ensuremath { \mathit { \hbar } } } }

\auenergy 925 \siunitx_declare_unit:Nn \aucharge { \ensuremath { \mathit { e } } } }

\aulength 926 \siunitx_declare_unit:Nx \auenergy

\aumass 927 {

\autime 928 \exp_not:N \ensuremath

\bohr 929 {

\hartree 930 \exp_not:N \mathit { E }

931 \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { h } } }

932 }

933 }

934 \siunitx_declare_unit:Nx \aulength

935 {

936 \exp_not:N \ensuremath

937 { \exp_not:N \mathit { a } } \c__siunitx_unit_math_subscript_tl { 0 } }

938 }

939 \siunitx_declare_unit:Nx \aumass

940 {

941 \exp_not:N \ensuremath

942 {

943 \exp_not:N \mathit { m } }

944 \c__siunitx_unit_math_subscript_tl { \exp_not:N \mathrm { e } } }

945 }

946 }

947 \siunitx_declare_unit:Nn \autime { \auaction \per \auenergy }

948 \siunitx_declare_unit:Nn \bohr { \aulength }

949 \siunitx_declare_unit:Nn \hartree { \auenergy }

(End definition for \auaction and others. These functions are documented on page 111.)

\angstrom There are then some day-to-day units which are accepted for use with SI, but are not

\bar part of the official specification.

\barn 950 \siunitx_declare_unit:Nx \angstrom

\bel 951 { _siunitx_unit_non_latin:n { "00C5 } }

\decibel 952 \siunitx_declare_unit:Nn \bar { bar }

\knot 953 \siunitx_declare_unit:Nn \barn { b }

\millimetremercury 954 \siunitx_declare_unit:Nn \bel { B }

\nauticalmile 955 \siunitx_declare_unit:Nn \decibel { \deci \bel }

\neper 956 \siunitx_declare_unit:Nn \knot { kn }

957 \siunitx_declare_unit:Nn \millimetremercury { mmHg }

958 \siunitx_declare_unit:Nn \nauticalmile { M }

959 \siunitx_declare_unit:Nn \neper { Np }

(End definition for \angstrom and others. These functions are documented on page 111.)

\dyne CGS units: similar to the set immediately above, these may be used for specific applica-

\erg tions.

\gal 960 \siunitx_declare_unit:Nn \dyne { dyn }

\gauss 961 \siunitx_declare_unit:Nn \erg { erg }

\maxwell 962 \siunitx_declare_unit:Nn \gal { Gal }

\oersted 963 \siunitx_declare_unit:Nn \gauss { G }

\phot 964 \siunitx_declare_unit:Nn \maxwell { Mx }

\poise 965 \siunitx_declare_unit:Nn \oersted { Oe }

\stilb 966 \siunitx_declare_unit:Nn \phot { ph }

\stokes

```

967 \siunitx_declare_unit:Nn \poise { P }
968 \siunitx_declare_unit:Nn \stilb { sb }
969 \siunitx_declare_unit:Nn \stokes { St }

```

(End definition for `\dyne` and others. These functions are documented on page 111.)

\percent For percent, the raw character is the most flexible way of handling output.

```

970 \siunitx_declare_unit:Nn \percent { \char "25 ~ }

```

(End definition for `\percent`. This function is documented on page 111.)

\square Basic powers.

\squared 971 \siunitx_declare_power:NNn \square \squared { 2 }

\cubic 972 \siunitx_declare_power:NNn \cubic \cubed { 3 }

\cubed

(End definition for `\square` and others. These functions are documented on page 111.)

6.12 Messages

```

973 \msg_new:nnnn { siunitx } { unit / dangling-part }
974 { Found~#1~part~with~no~unit. }
975 {
976   Each~#1~part~must~be~associated~with~a~unit:~a~#1~part~was~found~
977   but~no~following~unit~was~given.
978 }
979 \msg_new:nnnn { siunitx } { unit / duplicate-part }
980 { Duplicate~#1~part:~#2. }
981 {
982   Each~unit~may~have~only~one~#1:~\
983   the~additional~#1~part~'~#2'~will~be~ignored.
984 }
985 \msg_new:nnnn { siunitx } { unit / duplicate-sticky-per }
986 { Duplicate~\token_to_str:N \per. }
987 {
988   When~the~'sticky-per'~option~is~active,~only~one~
989   \token_to_str:N \per \ may~appear~in~a~unit.
990 }
991 \msg_new:nnnn { siunitx } { unit / literal }
992 { Literal~units~disabled. }
993 {
994   You~gave~the~literal~input~'~#1'~
995   but~literal~unit~output~is~disabled.
996 }
997 \msg_new:nnnn { siunitx } { unit / part-before-unit }
998 { Found~#1~part~before~first~unit:~#2. }
999 {
1000   The~#1~part~'~#2'~must~follow~after~a~unit:~
1001   it~cannot~appear~before~any~units~and~will~therefore~be~ignored.
1002 }

```

6.13 Standard settings for module options

Some of these follow naturally from the point of definition (*e.g.* boolean variables are always `false` to begin with), but for clarity everything is set here.

```

1003 \keys_set:nn { siunitx }

```

```

1004 {
1005   font-command          = \mathrm ,
1006   bracket-unit-denominator = true ,
1007   forbid-literal-units    = false ,
1008   fraction-command        = \frac ,
1009   inter-unit-product      = \, ,
1010   parse-units             = true ,
1011   per-mode                = power ,
1012   per-symbol              = / ,
1013   qualifier-mode          = subscript ,
1014   qualifier-phrase        = ,
1015   sticky-per              = false ,
1016   unit-close-bracket      = ) , % (
1017   unit-open-bracket       = ( % )
1018 }
1019 \end{package}

```

References

- [1] *The International System of Units (SI)*, <https://www.bipm.org/en/measurement-units/>.
- [2] *SI base units*, <https://www.bipm.org/en/publications/si-brochure/section2-1.html>.
- [3] *Units with special names and symbols; units that incorporate special names and symbols*, <https://www.bipm.org/en/publications/si-brochure/section2-2-2.html>.
- [4] *SI Prefixes*, <https://www.bipm.org/en/publications/si-brochure/chapter3.html>.
- [5] *Stating values of dimensionless quantities, or quantities of dimension one*, <https://www.bipm.org/en/publications/si-brochure/section5-3-7.html>.
- [6] *Non-SI units accepted for use with the International System of Units*, <https://www.bipm.org/en/publications/si-brochure/table6.html>.
- [7] *Non-SI units whose values in SI units must be obtained experimentally*, <https://www.bipm.org/en/publications/si-brochure/table7.html>.
- [8] *Other non-SI units*, <https://www.bipm.org/en/publications/si-brochure/table8.html>.
- [9] *Non-SI units associated with the CGS and the CGS-Gaussian system of units*, <https://www.bipm.org/en/publications/si-brochure/table9.html>.

Part X

siunitx-abbreviations – Abbreviations

<code>\A</code>	Abbreviations for currents.
<code>\pA</code>	
<code>\nA</code>	
<code>\uA</code>	
<code>\mA</code>	
<code>\kA</code>	

<code>\fg</code>	Abbreviations for masses.
<code>\pg</code>	
<code>\ng</code>	
<code>\ug</code>	
<code>\mg</code>	
<code>\g</code>	
<code>\kg</code>	
<code>\amu</code>	

<code>\K</code>	Abbreviations for temperature.
-----------------	--------------------------------

<code>\m</code>	Abbreviations for lengths.
<code>\pm</code>	
<code>\nm</code>	
<code>\um</code>	
<code>\mm</code>	
<code>\cm</code>	
<code>\dm</code>	
<code>\km</code>	

<code>\s</code>	Abbreviations for times.
<code>\as</code>	
<code>\fs</code>	
<code>\ps</code>	
<code>\ns</code>	
<code>\us</code>	
<code>\ms</code>	

<code>\Hz</code>	Abbreviations for frequencies.
<code>\mHz</code>	
<code>\kHz</code>	
<code>\MHz</code>	
<code>\GHz</code>	
<code>\THz</code>	

<code>\mol</code>	Abbreviations for moles.
<code>\fmol</code>	
<code>\pmol</code>	
<code>\nmol</code>	
<code>\umol</code>	
<code>\mmol</code>	
<code>\kmol</code>	

<code>\V</code>	Abbreviations for potentials.
<code>\pV</code>	
<code>\nV</code>	
<code>\uV</code>	
<code>\mV</code>	
<code>\kV</code>	

<code>\hl</code>	Abbreviations for volumes.
<code>\l</code>	
<code>\ml</code>	
<code>\ul</code>	
<code>\hL</code>	
<code>\L</code>	
<code>\mL</code>	
<code>\uL</code>	

<code>\W</code>	Abbreviations for powers.
<code>\uW</code>	
<code>\mW</code>	
<code>\kW</code>	
<code>\MW</code>	
<code>\GW</code>	

<code>\kJ</code>	Abbreviations for energies.
<code>\J</code>	
<code>\mJ</code>	
<code>\uJ</code>	
<code>\eV</code>	
<code>\meV</code>	
<code>\keV</code>	
<code>\MeV</code>	
<code>\GeV</code>	
<code>\TeV</code>	

<code>\N</code>	Abbreviations for forces.
<code>\mN</code>	
<code>\kN</code>	
<code>\MN</code>	

`\Pa` Abbreviations for pressures.
`\kPa`
`\MPa`
`\GPa`

`\mohm` Abbreviations for resistance.
`\kohm`
`\Mohm`

`\F` Abbreviations for capacitance.
`\fF`
`\pF`

`\dB` Abbreviation for decibel.

`\kWh` Abbreviation for kilowatt–hours.

1 siunitx-abbreviation implementation

Start the DocStrip guards.

```
1 <*package>
```

The abbreviation file contains a number of short (mainly two or three letter) versions of the usual long names. They are divided up into related groups, mainly to avoid an overly long list in one place.

\A Currents.

```
\pA 2 \siunitx_declare_unit:Nn \A { \ampere }
\nA 3 \siunitx_declare_unit:Nn \pA { \pico \ampere }
\uA 4 \siunitx_declare_unit:Nn \nA { \nano \ampere }
\mA 5 \siunitx_declare_unit:Nn \uA { \micro \ampere }
\kA 6 \siunitx_declare_unit:Nn \mA { \milli \ampere }
7 \siunitx_declare_unit:Nn \kA { \kilo \ampere }
```

(End definition for \A and others. These functions are documented on page 144.)

\Hz Then frequencies.

```
\mHz 8 \siunitx_declare_unit:Nn \Hz { \hertz }
\kHz 9 \siunitx_declare_unit:Nn \mHz { \milli \hertz }
\MHz 10 \siunitx_declare_unit:Nn \kHz { \kilo \hertz }
\GHz 11 \siunitx_declare_unit:Nn \MHz { \mega \hertz }
\THz 12 \siunitx_declare_unit:Nn \GHz { \giga \hertz }
13 \siunitx_declare_unit:Nn \THz { \tera \hertz }
```

(End definition for \Hz and others. These functions are documented on page 144.)

\mol Amounts of substance (moles).

```

\fmol 14 \siunitx_declare_unit:Nn \mol { \mole }
\pmol 15 \siunitx_declare_unit:Nn \fmol { \femto \mole }
\nmol 16 \siunitx_declare_unit:Nn \pmol { \pico \mole }
\umol 17 \siunitx_declare_unit:Nn \nmol { \nano \mole }
\mmol 18 \siunitx_declare_unit:Nn \umol { \micro \mole }
\kmol 19 \siunitx_declare_unit:Nn \mmol { \milli \mole }
20 \siunitx_declare_unit:Nn \kmol { \kilo \mole }

```

(End definition for \mol and others. These functions are documented on page 145.)

\V Potentials.

```

\pV 21 \siunitx_declare_unit:Nn \V { \volt }
\nV 22 \siunitx_declare_unit:Nn \pV { \pico \volt }
\uV 23 \siunitx_declare_unit:Nn \nV { \nano \volt }
\mV 24 \siunitx_declare_unit:Nn \uV { \micro \volt }
\kV 25 \siunitx_declare_unit:Nn \mV { \milli \volt }
26 \siunitx_declare_unit:Nn \kV { \kilo \volt }

```

(End definition for \V and others. These functions are documented on page 145.)

\hl Volumes.

```

\l 27 \siunitx_declare_unit:Nn \hl { \hecto \litre }
\ml 28 \siunitx_declare_unit:Nn \l { \litre }
\ul 29 \siunitx_declare_unit:Nn \ml { \milli \litre }
\hL 30 \siunitx_declare_unit:Nn \ul { \micro \litre }
\L 31 \siunitx_declare_unit:Nn \hL { \hecto \liter }
\mL 32 \siunitx_declare_unit:Nn \L { \liter }
\uL 33 \siunitx_declare_unit:Nn \mL { \milli \liter }
34 \siunitx_declare_unit:Nn \uL { \micro \liter }

```

(End definition for \hl and others. These functions are documented on page 145.)

\fg Masses.

```

\pg 35 \siunitx_declare_unit:Nn \fg { \femto \gram }
\ng 36 \siunitx_declare_unit:Nn \pg { \pico \gram }
\ug 37 \siunitx_declare_unit:Nn \ng { \nano \gram }
\mg 38 \siunitx_declare_unit:Nn \ug { \micro \gram }
\g 39 \siunitx_declare_unit:Nn \mg { \milli \gram }
\kg 40 \siunitx_declare_unit:Nn \g { \gram }
\amu 41 \siunitx_declare_unit:Nn \kg { \kilo \gram }
42 \siunitx_declare_unit:Nn \amu { \atomicmassunit }

```

(End definition for \fg and others. These functions are documented on page 144.)

\W Energies and powers

```

\uW 43 \siunitx_declare_unit:Nn \W { \watt }
\mW 44 \siunitx_declare_unit:Nn \uW { \micro \watt }
\kW 45 \siunitx_declare_unit:Nn \mW { \milli \watt }
\MW 46 \siunitx_declare_unit:Nn \kW { \kilo \watt }
\GW 47 \siunitx_declare_unit:Nn \MW { \mega \watt }
\kJ 48 \siunitx_declare_unit:Nn \GW { \giga \watt }
\J 49 \siunitx_declare_unit:Nn \J { \joule }
\mJ 50 \siunitx_declare_unit:Nn \uJ { \micro \joule }
\uJ 51 \siunitx_declare_unit:Nn \mJ { \milli \joule }

```

\eV
\meV
\keV
\MeV
\GeV
\TeV
\kWh

```

52 \siunitx_declare_unit:Nn \kJ { \kilo \joule }
53 \siunitx_declare_unit:Nn \eV { \electronvolt }
54 \siunitx_declare_unit:Nn \meV { \milli \electronvolt }
55 \siunitx_declare_unit:Nn \keV { \kilo \electronvolt }
56 \siunitx_declare_unit:Nn \MeV { \mega \electronvolt }
57 \siunitx_declare_unit:Nn \GeV { \giga \electronvolt }
58 \siunitx_declare_unit:Nn \TeV { \tera \electronvolt }
59 \siunitx_declare_unit:Nnn \kWh { \kilo \watt \hour }
60 { inter-unit-product = }

```

(End definition for \W and others. These functions are documented on page 145.)

\m Lengths.

```

\pm 61 \siunitx_declare_unit:Nn \m { \metre }
\nm 62 \siunitx_declare_unit:Nn \pm { \pico \metre }
\um 63 \siunitx_declare_unit:Nn \nm { \nano \metre }
\mm 64 \siunitx_declare_unit:Nn \um { \micro \metre }
\cm 65 \siunitx_declare_unit:Nn \mm { \milli \metre }
\dm 66 \siunitx_declare_unit:Nn \cm { \centi \metre }
\km 67 \siunitx_declare_unit:Nn \dm { \deci \metre }
68 \siunitx_declare_unit:Nn \km { \kilo \metre }

```

(End definition for \m and others. These functions are documented on page 144.)

\K Temperatures.

```

69 \siunitx_declare_unit:Nn \K { \kelvin }

```

(End definition for \K. This function is documented on page 144.)

\dB

```

70 \siunitx_declare_unit:Nn \dB { \deci \bel }

```

(End definition for \dB. This function is documented on page 146.)

\F Capacitance.

```

\fF 71 \siunitx_declare_unit:Nn \F { \farad }
\pF 72 \siunitx_declare_unit:Nn \fF { \femto \farad }
73 \siunitx_declare_unit:Nn \pF { \pico \farad }

```

(End definition for \F, \fF, and \pF. These functions are documented on page 146.)

\N Forces.

```

\mN 74 \siunitx_declare_unit:Nn \N { \newton }
\kN 75 \siunitx_declare_unit:Nn \mN { \milli \newton }
\mN 76 \siunitx_declare_unit:Nn \kN { \kilo \newton }
77 \siunitx_declare_unit:Nn \MN { \mega \newton }

```

(End definition for \N and others. These functions are documented on page 145.)

\Pa Pressures.

```

\kPa 78 \siunitx_declare_unit:Nn \Pa { \pascal }
\MPa 79 \siunitx_declare_unit:Nn \kPa { \kilo \pascal }
\GPa 80 \siunitx_declare_unit:Nn \MPa { \mega \pascal }
81 \siunitx_declare_unit:Nn \GPa { \giga \pascal }

```

(End definition for \Pa and others. These functions are documented on page 146.)

\mohm Resistances.

\kohm 82 \siunitx_declare_unit:Nn \mohm { \milli \ohm }
\Mohm 83 \siunitx_declare_unit:Nn \kohm { \kilo \ohm }
84 \siunitx_declare_unit:Nn \Mohm { \mega \ohm }

(End definition for \mohm, \kohm, and \Mohm. These functions are documented on page 146.)

\s Finally, times.

\as 85 \siunitx_declare_unit:Nn \s { \second }
\fs 86 \siunitx_declare_unit:Nn \as { \atto \second }
\ps 87 \siunitx_declare_unit:Nn \fs { \femto \second }
\ns 88 \siunitx_declare_unit:Nn \ps { \pico \second }
\us 89 \siunitx_declare_unit:Nn \ns { \nano \second }
\ms 90 \siunitx_declare_unit:Nn \us { \micro \second }
91 \siunitx_declare_unit:Nn \ms { \milli \second }

(End definition for \s and others. These functions are documented on page 144.)

92 </package>

Part XI

siunitx-command – Units as document command

1 Creating units as document commands

`\siunitx_command_create:` `\siunitx_command_create:`

Maps over the list of know unit commands and creates the appropriate document command to support them, as controlled by the options below.

1.1 Key-value options

The options defined by this submodule are available within the l3keys `siunitx` tree. These options are all preamble-only.

`free-standing-units` `free-standing-units = true|false`

`overwrite-commands` `overwrite-commands = true|false`

`space-before-unit` `space-before-unit = true|false`

`unit-optional-argument` `unit-optional-argument = true|false`

`use-xspace` `use-xspace = true|false`

2 siunitx-command implementation

Start the DocStrip guards.

1 `<*package>`

Identify the internal prefix (L^AT_EX3 DocStrip convention): only internal material in this *submodule* should be used directly.

2 `<@@=siunitx_command>`

`\l__siunitx_command_tmp_tl`

3 `\tl_new:N \l__siunitx_command_tmp_tl`

(End definition for `\l__siunitx_command_tmp_tl`.)

2.1 Options

```

\l_siunitx_command_create_bool
\l_siunitx_command_overwrite_bool
\l_siunitx_command_prespace_bool
\l_siunitx_command_optarg_bool
\l_siunitx_command_xspace_bool
4 \keys_define:nn { siunitx }
5   {
6     free-standing-units .bool_set:N =
7       \l__siunitx_command_create_bool ,
8     overwrite-commands .bool_set:N =
9       \l__siunitx_command_overwrite_bool ,
10    space-before-unit .bool_set:N =
11      \l__siunitx_command_prespace_bool ,
12    unit-optional-argument .bool_set:N =
13      \l__siunitx_command_optarg_bool ,
14    use-xspace .bool_set:N =
15      \l__siunitx_command_xspace_bool
16  }

```

(End definition for `\l__siunitx_command_create_bool` and others.)

These preamble-only options are all disabled at the start of the document.

```

17 \AtBeginDocument
18   {
19     \clist_map_inline:nn
20       {
21         free-standing-units ,
22         overwrite-commands ,
23         space-before-unit ,
24         unit-optional-argument ,
25         use-xspace
26       }
27     {
28       \keys_define:nn { siunitx }
29       {
30         #1 .code:n =
31           { \msg_warning:nnn { siunitx } { option-preamble-only } {#1} }
32       }
33     }
34   }
35 \msg_new:nnn { siunitx } { option-preamble-only }
36   { Option~'#1'~only~available~in~the~preamble. }

```

2.2 Creation of unit document commands

`\siunitx_command_create:` Creating document commands is all done by a single function which is set up using expansion: that way the tests are only run once. Other than that, this is all just a question of picking up all the various routes. Where the `soulpos` package is loaded *after* `siunitx`, the commands `\hl` and `\ul` will be created only after the hook is used. The `soul` package creates those using `\newcommand`, so we have to avoid an issue.

```

37 \cs_new_protected:Npn \siunitx_command_create:
38   {
39     \bool_if:NT \l__siunitx_command_create_bool
40     {
41       \__siunitx_command_create:
42       \@ifpackageloaded { soulpos }

```

```

43     {
44         \@ifpackageloaded { soul }
45         { }
46         {
47             \cs_undefine:N \hl
48             \cs_undefine:N \ul
49         }
50     }
51     { }
52 }

```

At the beginning of table cells and inside x-type expansion, all symbolic units need to have *some* definition.

```

53 \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
54 {
55     \cs_if_free:NT ##1
56     { \cs_set_protected:Npn ##1 { \ERROR } }
57 }
58 }
59 \AtBeginDocument { \siunitx_command_create: }
60 \cs_new_protected:Npn \__siunitx_command_create:
61 {
62     \bool_if:NT \l__siunitx_command_xspace_bool
63     { \RequirePackage { xspace } }
64     \bool_if:NT \l__siunitx_command_overwrite_bool
65     {
66         \seq_map_inline:Nn \l_siunitx_unit_symbolic_seq
67         { \cs_undefine:N ##1 }
68     }
69     \cs_set_protected:Npx \__siunitx_command_create:N ##1
70     {
71         \ProvideDocumentCommand ##1 { \bool_if:NT \l__siunitx_command_optarg_bool { o } }
72         {
73             \mode_leave_vertical:
74             \group_begin:
75             \bool_if:NTF \l__siunitx_command_optarg_bool
76             { \exp_not:N \IfNoValueTF {####1} }
77             { \use_i:nn }
78             {
79                 \siunitx_unit_options_apply:n {##1}
80                 \bool_if:NT \l__siunitx_command_prespace_bool { \exp_not:N \ }
81                 \siunitx_unit_format:nN {##1}
82                 \exp_not:N \l__siunitx_command_tmp_tl
83                 \siunitx_print:nV { unit }
84                 \exp_not:N \l__siunitx_command_tmp_tl
85             }
86             { \siunitx_quantity:nn {####1} {##1} }
87             \group_end:
88             \bool_if:NT \l__siunitx_command_xspace_bool { \exp_not:N \xspace }
89         }
90     }
91     \seq_map_function:NN \l_siunitx_unit_seq \__siunitx_command_create:N
92 }
93 \cs_new_protected:Npn \__siunitx_command_create:N #1 { }

```

(End definition for `\siunitx_command_create:`, `_siunitx_command_create:`, and `_siunitx_command_create:N`. This function is documented on page [150](#).)

2.3 Standard settings for module options

Some of these follow naturally from the point of definition (*e.g.* boolean variables are always `false` to begin with), but for clarity everything is set here.

```
94 \keys_set:nn { siunitx }
95   {
96     free-standing-units    = false ,
97     overwrite-commands    = false ,
98     space-before-unit     = false ,
99     unit-optional-argument = false ,
100    use-xspace              = false
101  }
102 \end{package}
```

Part XII

siunitx-emulation – Emulation

1 siunitx-emulation implementation

Identify the internal prefix (L^AT_EX3 DocStrip convention). In contrast to other parts of the bundle, the functions here may need to redefine those from various submodules.

```
1 <@@=siunitx>
```

1.1 Version 2

Start the DocStrip guards.

```
2 <*package>
   Some messages.
3 \msg_new:nnn { siunitx } { option-deprecated }
4 {
5   Option~"#1"~has~been~deprecated~in~this~release.\\ \\
6   Use~"#2"~as~a~replacement.
7 }
8 \msg_new:nnn { siunitx } { option-removed }
9 { Option~"#1"~has~been~removed~in~this~release. }
```

```
\_siunitx_option_deprecated:nn
\_siunitx_option_deprecated:nnn
\_siunitx_option_deprecated:nnV
```

Abstract out a simple wrapper.

```
10 \cs_new_protected:Npn \_siunitx_option_deprecated:nn #1#2
11 {
12   \msg_info:nnnn { siunitx } { option-deprecated } {#1} {#2}
13   \keys_set:nn { siunitx } {#2}
14 }
15 \cs_new_protected:Npn \_siunitx_option_deprecated:nnn #1#2#3
16 {
17   \msg_info:nnnn { siunitx } { option-deprecated } {#1} {#2}
18   \keys_set:nn { siunitx } { #2 = #3 }
19 }
20 \cs_generate_variant:Nn \_siunitx_option_deprecated:nnn { nnV }
```

(End definition for _siunitx_option_deprecated:nn and _siunitx_option_deprecated:nnn.)

1.2 Preamble commands

1.3 Document commands

```
\DeclareSIPrePower
\DeclareSIPostPower
```

Simply use a throw-away command for the part we do not need: this can be followed by some clean-up

```
21 \NewDocumentCommand \DeclareSIPrePower { +m m }
22 {
23   \siunitx_declare_power:NNn #1 \_siunitx_tmp:w {#2}
24   \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
25 }
26 \NewDocumentCommand \DeclareSIPostPower { +m m }
27 {
```

```

28 \siunitx_declare_power:NNn \_siunitx_tmp:w #1 {#2}
29 \seq_remove_all:Nn \l_siunitx_unit_symbolic_seq { \_siunitx_tmp:w }
30 }

```

(End definition for `\DeclareSIPrePower` and `\DeclareSIPostPower`. These functions are documented on page ??.)

`\si` A straight copy of `\unit`.

```

31 \NewDocumentCommand \si { 0 { } m }
32 {
33   \mode_leave_vertical:
34   \group_begin:
35     \keys_set:nn { siunitx } {#1}
36     \siunitx_unit_format:nN {#2} \l__siunitx_tmp_tl
37     \siunitx_print:nV { unit } \l__siunitx_tmp_tl
38   \group_end:
39 }

```

(End definition for `\si`. This function is documented on page ??.)

`\SI` Almost the same as `\qty`, but with the addition pre-unit.

```

40 \NewDocumentCommand \SI { 0 { } m o m }
41 {
42   \mode_leave_vertical:
43   \group_begin:
44     \keys_set:nn { siunitx } {#1}
45     \IfNoValueF {#3}
46     {
47       \siunitx_unit_format:nN {#3} \l__siunitx_tmp_tl
48       \siunitx_print:nV { unit } \l__siunitx_tmp_tl
49       \nobreak
50     }
51     \siunitx_quantity:nn {#2} {#4}
52   \group_end:
53 }

```

(End definition for `\SI`. This function is documented on page ??.)

1.4 Angle options

All straight-forward emulation.

```

54 \keys_define:nn { siunitx }
55 {
56   add-arc-degree-zero .code:n =
57   {
58     \__siunitx_option_deprecated:nnV
59     { add-arc-degree-zero }
60     { fill-arc-degrees }
61     \l_keys_value_tl
62   } ,
63   add-arc-degree-zero .default:n = true ,
64   add-arc-minute-zero .code:n =
65   {
66     \__siunitx_option_deprecated:nnV

```

```

67         { add-arc-minute-zero }
68         { fill-arc-minutes }
69         \l_keys_value_tl
70     } ,
71     add-arc-minute-zero .default:n = true ,
72     add-arc-second-zero .code:n =
73     {
74         \__siunitx_option_deprecated:nnV
75         { add-arc-second-zero }
76         { fill-arc-seconds }
77         \l_keys_value_tl
78     } ,
79     add-arc-second-zero .default:n = true
80 }

```

1.5 Combination functions options

```

81 \keys_define:nn { siunitx }
82 {
83     list-mode / brackets .code:n =
84     {
85         \__siunitx_option_deprecated:nn
86         { list-mode~~brackets }
87         { list-mode~~bracket }
88     } ,
89     range-mode / brackets .code:n =
90     {
91         \__siunitx_option_deprecated:nn
92         { range-mode~~brackets }
93         { range-mode~~bracket }
94     }
95 }

```

1.6 Command options

```

96 \keys_define:nn { siunitx }
97 {
98     overwrite-functions .code:n =
99     {
100         \__siunitx_option_deprecated:nnV
101         { overwrite-functions }
102         { overwrite-commands }
103         \l_keys_value_tl
104     } ,
105     overwrite-functions .default:n = true
106 }

```

1.7 Print options

At present, simply remove these.

```

107 \keys_define:nn { siunitx }
108 {
109     detect-all .code:n =
110     {
111         \msg_warning:nnn { siunitx } { option-removed }

```



```

112         { detect-all }
113     } ,
114     detect-display-math .code:n =
115     {
116         \msg_warning:nnn { siunitx } { option-removed }
117         { detect-display-math }
118     } ,
119     detect-family .code:n =
120     {
121         \msg_warning:nnn { siunitx } { option-removed }
122         { detect-family }
123     } ,
124     detect-inline-family .code:n =
125     {
126         \msg_warning:nnn { siunitx } { option-removed }
127         { detect-inline-family }
128     } ,
129     detect-inline-weight .code:n =
130     {
131         \msg_warning:nnn { siunitx } { option-removed }
132         { detect-inline-weight }
133     } ,
134     detect-mode .code:n =
135     {
136         \msg_warning:nnn { siunitx } { option-removed }
137         { detect-mode }
138     } ,
139     detect-none .code:n =
140     {
141         \msg_warning:nnn { siunitx } { option-removed }
142         { detect-none }
143     } ,
144     detect-shape .code:n =
145     {
146         \msg_warning:nnn { siunitx } { option-removed }
147         { detect-shape }
148     } ,
149     detect-weight .code:n =
150     {
151         \msg_warning:nnn { siunitx } { option-removed }
152         { detect-weight }
153     }
154 }

```

The old font insertion options: all removed.

```

155 \keys_define:nn { siunitx }
156 {
157     math-rm .code:n =
158     {
159         \msg_warning:nnn { siunitx } { option-removed }
160         { math-rm }
161     } ,
162     math-sf .code:n =
163     {
164         \msg_warning:nnn { siunitx } { option-removed }

```

```

165         { math-sf }
166     } ,
167     math-tt .code:n =
168     {
169         \msg_warning:nnn { siunitx } { option-removed }
170         { math-tt }
171     } ,
172     number-math-rm .code:n =
173     {
174         \msg_warning:nnn { siunitx } { option-removed }
175         { number-math-rm }
176     } ,
177     number-math-sf .code:n =
178     {
179         \msg_warning:nnn { siunitx } { option-removed }
180         { number-math-sf }
181     } ,
182     number-math-tt .code:n =
183     {
184         \msg_warning:nnn { siunitx } { option-removed }
185         { number-math-tt }
186     } ,
187     number-text-rm .code:n =
188     {
189         \msg_warning:nnn { siunitx } { option-removed }
190         { number-text-rm }
191     } ,
192     number-text-sf .code:n =
193     {
194         \msg_warning:nnn { siunitx } { option-removed }
195         { number-text-sf }
196     } ,
197     number-text-tt .code:n =
198     {
199         \msg_warning:nnn { siunitx } { option-removed }
200         { number-text-tt }
201     } ,
202     text-rm .code:n =
203     {
204         \msg_warning:nnn { siunitx } { option-removed }
205         { text-rm }
206     } ,
207     text-sf .code:n =
208     {
209         \msg_warning:nnn { siunitx } { option-removed }
210         { text-sf }
211     } ,
212     text-tt .code:n =
213     {
214         \msg_warning:nnn { siunitx } { option-removed }
215         { text-tt }
216     } ,
217     unit-math-rm .code:n =
218     {

```

```

219         \msg_warning:nnn { siunitx } { option-removed }
220         { unit-math-rm }
221     } ,
222     unit-math-sf .code:n =
223     {
224         \msg_warning:nnn { siunitx } { option-removed }
225         { unit-math-sf }
226     } ,
227     unit-math-tt .code:n =
228     {
229         \msg_warning:nnn { siunitx } { option-removed }
230         { unit-math-tt }
231     } ,
232     unit-text-rm .code:n =
233     {
234         \msg_warning:nnn { siunitx } { option-removed }
235         { unit-text-rm }
236     } ,
237     unit-text-sf .code:n =
238     {
239         \msg_warning:nnn { siunitx } { option-removed }
240         { unit-text-sf }
241     } ,
242     unit-text-tt .code:n =
243     {
244         \msg_warning:nnn { siunitx } { option-removed }
245         { unit-text-tt }
246     }
247 }

```

1.8 Number options

For the basic emulation, just set up some information.

```

248 \keys_define:nn { siunitx }
249 {
250     input-protect-tokens .code:n =
251     {
252         \msg_warning:nnn { siunitx } { option-removed }
253         { input-protect-tokens }
254     }
255 }

```

Options for number processing: largely removals.

```

256 \keys_define:nn { siunitx }
257 {
258     add-decimal-zero .choice: ,
259     add-decimal-zero / false .code:n =
260     {
261         \__siunitx_option_deprecated:nn
262         { add-decimal-zero }
263         { minimum-decimal-digits-=-0 }
264     } ,
265     add-decimal-zero / true .code:n =
266     {

```

```

267     \__siunitx_option_deprecated:nn
268     { add-decimal-zero }
269     { minimum-decimal-digits~==1 }
270   } ,
271   add-integer-zero .code:n =
272   {
273     \msg_warning:nnn { siunitx } { option-removed }
274     { add-integer-zero }
275   } ,
276   close-bracket .code:n =
277   {
278     \__siunitx_option_deprecated:nnV
279     { close-bracket }
280     { number-close-bracket }
281     \l_keys_value_tl
282   } ,
283   explicit-sign .code:n =
284   {
285     \str_if_eq:nnTF {#1} { + }
286     {
287       \__siunitx_option_deprecated:nn
288       { explicit-sign }
289       { print-implicit-plus~==true }
290     }
291     {
292       \msg_warning:nnn { siunitx } { option-removed }
293       { explicit-sign }
294     }
295   } ,
296   omit-uncertainty .code:n =
297   {
298     \__siunitx_option_deprecated:nnV
299     { omit-uncertainty }
300     { drop-uncertainty }
301     \l_keys_value_tl
302   } ,
303   omit-uncertainty .default:n = true ,
304   retain-explicit-plus .code:n =
305   {
306     \__siunitx_option_deprecated:nnV
307     { retain-explicit-plus }
308     { track-explicit-plus }
309     \l_keys_value_tl
310   } ,
311   open-bracket .code:n =
312   {
313     \__siunitx_option_deprecated:nnV
314     { open-bracket }
315     { number-open-bracket }
316     \l_keys_value_tl
317   } ,
318   retain-explicit-plus .default:n = true ,
319   retain-unity-mantissa .code:n =
320   {

```

```

321     \_siunitx_option_deprecated:nnV
322     { retain-unity-mantissa }
323     { print-unity-mantissa }
324     \l_keys_value_tl
325   } ,
326   retain-unity-mantissa .default:n = true ,
327   retain-zero-exponent .code:n =
328   {
329     \_siunitx_option_deprecated:nnV
330     { retain-zero-exponent }
331     { print-zero-exponent }
332     \l_keys_value_tl
333   } ,
334   retain-zero-exponent .default:n = true ,
335   scientific-notation .choice: ,
336   scientific-notation / engineering .code:n =
337   {
338     \_siunitx_option_deprecated:nn
339     { scientific-notation~~engineering }
340     { exponent-mode~~engineering }
341   } ,
342   scientific-notation / fixed .code:n =
343   {
344     \_siunitx_option_deprecated:nn
345     { scientific-notation~~fixed }
346     { exponent-mode~~fixed }
347   } ,
348   scientific-notation / false .code:n =
349   {
350     \_siunitx_option_deprecated:nn
351     { scientific-notation~~false }
352     { exponent-mode~~input }
353   } ,
354   scientific-notation / true .code:n =
355   {
356     \_siunitx_option_deprecated:nn
357     { scientific-notation~~true }
358     { exponent-mode~~scientific }
359   } ,
360   zero-decimal-to-integer .code:n =
361   {
362     \_siunitx_option_deprecated:nnV
363     { zero-decimal-to-integer }
364     { drop-zero-decimal }
365     \l_keys_value_tl
366   } ,
367   zero-decimal-to-integer .default:n = true
368 }

```

1.8.1 Table options

All straight-forward emulation.

```

369 \keys_define:nn { siunitx }
370 {

```

```

371 table-align-text-post .code:n =
372 {
373     \__siunitx_option_deprecated:nnV
374     { table-align-text-post }
375     { table-align-text-after }
376     \l_keys_value_tl
377 },
378 table-align-text-post .default:n = true ,
379 table-align-text-pre .code:n =
380 {
381     \__siunitx_option_deprecated:nnV
382     { table-align-text-pre }
383     { table-align-text-before }
384     \l_keys_value_tl
385 },
386 table-align-text-pre .default:n = true ,
387 table-number-alignment / center-decimal-marker .code:n =
388 {
389     \msg_info:nnnn { siunitx } { option-deprecated }
390     { table-number-alignment~==center-decimal-marker }
391     { table-alignment-mode~==marker }
392     \keys_set:nn
393     { siunitx }
394     { table-alignment-mode = marker }
395 },
396 table-omit-exponent .code:n =
397 {
398     \__siunitx_option_deprecated:nnV
399     { table-omit-exponent }
400     { drop-exponent }
401     \l_keys_value_tl
402 },
403 table-omit-exponent .default:n = true ,
404 table-parse-only .code:n =
405 {
406     \msg_info:nnnn { siunitx } { option-deprecated }
407     { table-parse-only }
408     { table-alignment-mode~==none }
409     \str_if_eq:VnTF \l_keys_value_tl { false }
410     {
411         \keys_set:nn
412         { siunitx }
413         { table-alignment-mode = marker }
414     }
415     {
416         \keys_set:nn
417         { siunitx }
418         { table-alignment-mode = none }
419     }
420 },
421 table-space-text-post .code:n =
422 {
423     \msg_info:nnnn { siunitx } { option-deprecated }
424     { table-space-text-post }

```

```

425         { table-format }
426         \tl_set:Nn \l__siunitx_table_after_model_tl {#1}
427     } ,
428     table-space-text-pre .code:n =
429     {
430         \msg_info:nnnn { siunitx } { option-deprecated }
431         { table-space-text-post }
432         { table-format }
433         \tl_set:Nn \l__siunitx_table_before_model_tl {#1}
434     }
435 }

\__siunitx_option_table_format:n
\__siunitx_option_table_comparator:nnnnnnn
436 \cs_new_protected:Npn \__siunitx_option_table_format:n #1
437 {
438     \msg_info:nnnn { siunitx } { option-deprecated }
439     { table- #1 }
440     { table-format }
441     \tl_set:Nx \l__siunitx_table_format_tl
442     {
443         \cs:w __siunitx_option_table_ #1 :nnnnnnnn
444         \exp_after:wN \exp_after:wN \exp_after:wN \cs_end:
445         \exp_after:wN \l__siunitx_table_format_tl
446         \exp_after:wN { \l_keys_value_tl }
447     }
448     \exp_after:wN \__siunitx_table_generate_model:nnnnnnn
449     \l__siunitx_table_format_tl
450 }
451 \cs_new:Npn \__siunitx_option_table_comparator:nnnnnnnn #1#2#3#4#5#6#7#8
452 { \exp_not:n { {#8} {#2} {#3} {#4} {#5} {#6} {#7} } }
453 \cs_new:cpn { __siunitx_option_table_figures-decimal:nnnnnnnn }
454 #1#2#3#4#5#6#7#8
455 { \exp_not:n { {#1} {#2} {#3} {#8} {#5} {#6} {#7} } }
456 \cs_new:cpn { __siunitx_option_table_figures-exponent:nnnnnnnn }
457 #1#2#3#4#5#6#7#8
458 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#6} {#8} } }
459 \cs_new:cpn { __siunitx_option_table_figures-integer:nnnnnnnn }
460 #1#2#3#4#5#6#7#8
461 { \exp_not:n { {#1} {#2} {#8} {#4} {#5} {#6} {#7} } }
462 \cs_new:cpn { __siunitx_option_table_figures-uncertainty:nnnnnnnn }
463 #1#2#3#4#5#6#7#8
464 { \exp_not:n { {#1} {#2} {#3} {#4} { { S } } {#8} } {#6} {#7} } }
465 \cs_new:cpn { __siunitx_option_table_sign-exponent:nnnnnnnn }
466 #1#2#3#4#5#6#7#8
467 { \exp_not:n { {#1} {#2} {#3} {#4} {#5} {#8} {#7} } }
468 \cs_new:cpn { __siunitx_option_table_sign-mantissa:nnnnnnnn }
469 #1#2#3#4#5#6#7#8
470 { \exp_not:n { {#1} {#8} {#3} {#4} {#5} {#6} {#7} } }

(End definition for \__siunitx_option_table_format:n and others.)
Options which all use the same emulation set up.
471 \keys_define:nn { siunitx }
472 {
473     table-comparator .code:n =

```

```

474 { \_siunitx_option_table_format:n { comparator } } ,
475 table-figures-decimal .code:n =
476 { \_siunitx_option_table_format:n { figures-decimal } } ,
477 table-figures-exponent .code:n =
478 { \_siunitx_option_table_format:n { figures-exponent } } ,
479 table-figures-integer .code:n =
480 { \_siunitx_option_table_format:n { figures-integer } } ,
481 table-figures-uncertainty .code:n =
482 { \_siunitx_option_table_format:n { figures-uncertainty } } ,
483 table-sign-exponent .code:n =
484 { \_siunitx_option_table_format:n { sign-exponent } } ,
485 table-sign-mantissa .code:n =
486 { \_siunitx_option_table_format:n { sign-mantissa } }
487 }

```

1.9 Unit options

```

488 \keys_define:nn { siunitx }
489 {
490   allow-number-unit-breaks .code:n =
491   {
492     \_siunitx_option_deprecated:nnV
493     { allow-number-unit-breaks }
494     { allow-quantity-breaks }
495     \l_keys_value_tl
496   } ,
497   allow-number-unit-breaks .default:n = true ,
498   fraction-function .code:n =
499   {
500     \_siunitx_option_deprecated:nnV
501     { fraction-function }
502     { fraction-command }
503     \l_keys_value_tl
504   } ,
505   literal-superscript-as-power .code:n =
506   {
507     \msg_warning:nnn { siunitx } { option-removed }
508     { literal-superscript-as-power }
509   } ,
510   number-unit-product .code:n =
511   {
512     \_siunitx_option_deprecated:nnV
513     { number-unit-product }
514     { quantity-product }
515     \l_keys_value_tl
516   } ,
517   per-mode / reciprocal .code:n =
518   {
519     \_siunitx_option_deprecated:nn
520     { per-mode-==reciprocal }
521     { per-mode-==power }
522   } ,
523   per-mode / reciprocal-positive-first .code:n =
524   {

```



```

525     \__siunitx_option_deprecated:nn
526     { per-mode~==reciprocal-positive-first }
527     { per-mode~==power-positive-first }
528   } ,
529   power-font .code:n =
530   {
531     \msg_warning:nnn { siunitx } { option-removed }
532     { power-font }
533   } ,
534   prefixes-as-symbols .choice: ,
535   prefixes-as-symbols / false . code:n =
536   {
537     \__siunitx_option_deprecated:nn
538     { prefixes-as-symbols~==false }
539     { prefix-mode~==power }
540   } ,
541   prefixes-as-symbols / true . code:n =
542   {
543     \__siunitx_option_deprecated:nn
544     { prefixes-as-symbols~==true }
545     { prefix-mode~==symbol }
546   } ,
547   qualifier-mode / brackets .code:n =
548   {
549     \__siunitx_option_deprecated:nn
550     { qualifier-mode~==brackets }
551     { qualifier-mode~==bracket }
552   } ,
553   qualifier-mode / space .code:n =
554   {
555     \msg_info:nnnn { siunitx } { option-deprecated }
556     { qualifier-mode~==space }
557     { qualifier-mode~==phrase"~plus~"qualifier-phrase=\ }
558     \keys_set:nn
559     { siunitx }
560     { qualifier-mode = phrase, qualifier-phrase = \ }
561   } ,
562   qualifier-mode / text .code:n =
563   {
564     \__siunitx_option_deprecated:nn
565     { qualifier-mode~==text }
566     { qualifier-mode~==combine }
567   }
568 }
569 \end{package}

```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

\, *71, 106, 1009, 1654, 1660*
 \- *340*
 \\ *5, 982*
 _ *8, 211, 222, 290, 367*
 \~ *175*

_ *80, 113, 557, 560, 989*

A

\A *2, 144*
 allow-quantity-breaks *78*
 \ampere *2, 3, 4, 5, 6, 7, 109, 841*
 \amu *35, 144*
 \ang *6, 67, 164*
 \angInBookmark *157*
 angle-mode *8*
 \angstrom *82, 87, 111, 950*
 \approx *1656*
 arc-separator *8*
 arc-separator-over-decimal *8*
 \arcminute *110, 169, 901*
 \arcsecond *110, 170, 901*
 \as *85, 144*
 \astronomicalunit *111, 904*
 \AtBeginDocument
 *5, 5, 17, 28, 33, 59, 64, 80,*
 123, 150, 175, 200, 214, 226, 235, 306
 \atomicmassunit *42, 111, 904*
 \atto *86, 109, 850*
 \auaction *111, 924*
 \aucharge *111, 924*
 \auenergy *111, 924*
 \aulength *111, 924*
 \aumass *111, 924*
 \autime *111, 924*

B

\bar *111, 950*
 \barn *111, 950*
 \becquerel *110, 871*
 \begin *193*
 \bel *70, 111, 950*
 \bfseries *65*
 \bohr *111, 924*
 \boldmath *66*

bool commands:

\bool_if:NTF ... *10, 16, 17, 18, 32,*
 34, 34, 39, 41, 55, 62, 64, 71, 75, 78,
 80, 88, 88, 90, 122, 125, 138, 146,
 146, 155, 169, 177, 184, 191, 246,
 253, 258, 263, 344, 377, 379, 388,
 404, 412, 434, 473, 474, 500, 518,
 526, 545, 551, 574, 590, 599, 669,
 681, 689, 701, 718, 738, 752, 761,
 764, 823, 833, 1108, 1236, 1343,
 1351, 1360, 1408, 1486, 1555, 1566
 \bool_lazy_all:nTF *1383*
 \bool_lazy_all_p:n *919, 937*
 \bool_lazy_and:nnTF *104,*
 184, 428, 556, 574, 615, 703, 711, 1550
 \bool_lazy_or:nnTF *13,*
 24, 24, 41, 98, 114, 198, 283, 334,
 440, 503, 820, 916, 934, 1196, 1546
 \bool_new:N *3, 5, 6, 6, 7, 9, 10, 37, 38,*
 75, 83, 84, 87, 295, 506, 507, 508,
 509, 510, 511, 512, 514, 599, 1312, 1313
 \bool_set_false:N *9, 12, 12, 16, 17,*
 21, 21, 22, 26, 27, 92, 96, 101, 102,
 131, 179, 199, 286, 301, 302, 454,
 455, 461, 462, 463, 464, 468, 469,
 470, 475, 478, 482, 530, 532, 573,
 588, 595, 685, 1270, 1274, 1279, 1280
 \bool_set_true:N *11,*
 14, 14, 16, 17, 22, 91, 97, 101, 136,
 198, 225, 288, 300, 390, 456, 457,
 471, 476, 477, 483, 484, 485, 489,
 490, 491, 492, 533, 586, 675, 706,
 745, 1264, 1265, 1269, 1275, 1581, 1582
 \c_false_bool *47, 120, 384, 446, 450,*
 455, 457, 464, 488, 494, 508, 512, 527
 \c_true_bool *44, 117,*
 369, 384, 444, 457, 461, 488, 494, 508

box commands:

\box_clear:N *440, 489*
 \box_new:N
 ... *3, 12, 164, 165, 229, 230, 416, 417*
 \box_use:N *239*
 \box_use_drop:N
 ... *234, 236, 365, 366, 410, 411,*
 463, 470, 480, 481, 533, 534, 535, 536
 \box_wd:N *218, 219, 238, 239,*
 242, 250, 253, 342, 381, 385, 399,
 438, 445, 446, 449, 457, 487, 493,

524, 548, 559, 560, 561, 576, 580,
581, 594, 595, 605, 613, 614, 616,
633, 634, 655, 666, 685, 687, 697, 709
bracket-negative-numbers 22
bracket-unit-denominator 112

C

\cancel 106, 106, 112, 118
\candela 109, 841
\centi 66, 109, 850
\char 970
char commands:
 \char_codepoint_to_bytes:n .. 52, 831
 \char_generate:nn 8, 46,
 57, 59, 211, 222, 290, 367, 825, 836, 838
 \char_set_active_eq:NN
 185, 187, 189, 348, 389
 \char_set_catcode_active:N 340
 \char_set_catcode_active:n 175
 \char_set_mathcode:nn 349, 390
 \char_value_catcode:n 46, 825
\circ 73, 873, 903
clist commands:
 \clist_map_break: 113
 \clist_map_function:nN 30, 35
 \clist_map_inline:nn
 19, 101, 116, 117, 118, 433, 535
\cm 61, 144
color 65
\coulomb 110, 871
\cr 28, 88
cs commands:
 \cs:w 111, 443, 630, 845
 \cs_end: 111, 444, 633, 848
 \cs_generate_variant:Nn
 3, 3, 4, 5, 5, 6,
 11, 20, 28, 58, 61, 72, 100, 160, 164,
 170, 172, 174, 219, 228, 434, 694,
 735, 858, 1020, 1023, 1034, 1405, 1537
 \cs_if_eq:NNTF 356
 \cs_if_exist:NTF 34, 134, 137
 \cs_if_free:NTF 7, 17, 55, 125
 \cs_new:Npn . 45, 49, 54, 59, 61, 66,
 132, 142, 162, 172, 173, 196, 203,
 209, 210, 232, 321, 360, 451, 453,
 456, 459, 462, 465, 468, 539, 549,
 567, 586, 641, 650, 652, 672, 684,
 695, 701, 707, 713, 719, 721, 737,
 744, 764, 769, 780, 788, 802, 815,
 824, 828, 829, 833, 839, 851, 853,
 859, 868, 884, 898, 909, 930, 948,
 978, 1005, 1012, 1019, 1021, 1024,
 1035, 1045, 1052, 1059, 1060, 1061,
 1068, 1070, 1076, 1081, 1095, 1103,

1112, 1127, 1135, 1153, 1166, 1167,
1185, 1194, 1211, 1220, 1242, 1314,
1316, 1318, 1323, 1333, 1339, 1358,
1364, 1373, 1381, 1392, 1406, 1417,
1426, 1428, 1430, 1432, 1438, 1448,
1453, 1473, 1482, 1484, 1503, 1523,
1538, 1543, 1575, 1610, 1616, 1621
\cs_new:Npx 208
\cs_new_eq:NN .. 22, 130, 204, 231, 670
\cs_new_protected:Npn
 7, 10, 14, 14, 15,
 20, 20, 21, 23, 24, 25, 27, 30, 36, 37,
 38, 38, 39, 40, 44, 49, 49, 49, 51, 53,
 55, 59, 60, 62, 64, 70, 73, 73, 75, 76,
 82, 85, 86, 88, 88, 93, 95, 100, 101,
 104, 105, 107, 120, 122, 129, 131,
 133, 134, 134, 137, 139, 139, 144,
 144, 150, 150, 153, 157, 160, 161,
 165, 165, 166, 167, 171, 171, 173,
 176, 182, 183, 185, 187, 187, 189,
 194, 196, 198, 209, 211, 214, 223,
 223, 225, 225, 227, 232, 235, 244,
 245, 249, 251, 259, 260, 262, 265,
 271, 273, 281, 284, 285, 295, 297,
 298, 299, 303, 310, 313, 316, 322,
 323, 324, 325, 327, 330, 332, 335,
 337, 345, 346, 347, 352, 353, 354,
 360, 365, 367, 368, 372, 373, 373,
 375, 386, 394, 395, 400, 403, 414,
 415, 426, 431, 432, 435, 436, 438,
 448, 453, 459, 466, 472, 478, 483,
 497, 520, 535, 540, 541, 541, 546,
 551, 559, 569, 570, 571, 588, 597,
 602, 603, 613, 618, 620, 626, 626,
 629, 638, 640, 641, 646, 650, 651,
 660, 663, 671, 677, 679, 682, 694,
 699, 713, 725, 736, 748, 757, 769,
 785, 794, 797, 803, 821, 831, 841, 1234
\cs_new_protected:Npx
 30, 176, 205, 220, 279, 341, 360
\cs_set:Npn 30, 35
\cs_set:Npx 157, 159
\cs_set_eq:NN 143, 156, 182, 288
\cs_set_protected:Npn
 28, 56, 68, 201, 207,
 218, 218, 239, 243, 310, 324, 334, 391
\cs_set_protected:Npx 69, 161
\cs_to_str:N 160, 182
\cs_undefine:N 47, 48, 67
\cubed 111, 971
\cubic 111, 971

D
\dalton 111, 904

<code>\day</code>	110, 894	448, 468, 480, 507, 520, 548, 562,	
<code>\dB</code>	70, 146	578, 607, 632, 638, 800, 826, 835,	
<code>\deca</code>	109, 860	836, 837, 847, 1090, 1239, 1321, 1618	
<code>\deci</code>	67, 70, 109, 850, 955	<code>\exp_args:Nc</code>	123, 159
<code>\decibel</code>	111, 950	<code>\exp_args:Ne</code>	42, 56, 57
<code>\DeclareExpandableDocumentCommand</code> .	159	<code>\exp_args:Nf</code>	644, 893, 904, 1150, 1491
<code>\DeclareFontEncoding</code>	9	<code>\exp_args:NNc</code>	204
<code>\DeclareFontSubstitution</code>	10	<code>\exp_args:Nnno</code>	1204
<code>\DeclareSIPostPower</code>	21	<code>\exp_args:NNNV</code>	25, 97, 202, 291, 512, 773
<code>\DeclareSIPower</code>	40	<code>\exp_args:NNnx</code>	242
<code>\DeclareSIPrefix</code>	40	<code>\exp_args:NNV</code>	16, 351
<code>\DeclareSIPrePower</code>	21	<code>\exp_args:NV</code> ..	43, 108, 244, 272, 1048
<code>\DeclareSIQualifier</code>	40	<code>\exp_args:Nv</code>	90
<code>\DeclareSIUnit</code>	40	<code>\exp_args:Nx</code>	502
<code>\DeclareTextSymbol</code>	20, 128	<code>\exp_last_unbraced:Nf</code>	
<code>\DeclareTextSymbolDefault</code>	19, 127	51, 830, 986, 1213
<code>\degree</code>	73, 80, 110, 161, 168, 901	<code>\exp_last_unbraced:NV</code>	319
<code>\degreeCelsius</code>	110, 163, 871	<code>\exp_not:N</code>	32,
<code>\deka</code>	109, 860	35, 36, 57, 59, 67, 70, 76, 78, 80, 82,	
dim commands:		84, 88, 118, 163, 165, 180, 181, 183,	
<code>\dim_add:Nn</code>	706	186, 188, 189, 190, 208, 209, 210,	
<code>\dim_compare:nNnTF</code>		212, 213, 215, 216, 223, 232, 233,	
.....	218, 237, 438, 444, 487	235, 239, 243, 244, 245, 247, 248,	
<code>\dim_compare_p:nNn</code>	576	251, 253, 254, 255, 255, 256, 257,	
<code>\dim_new:N</code>	4, 4, 418	258, 260, 261, 263, 265, 266, 267,	
<code>\dim_set:Nn</code>	253, 604, 631, 655, 666, 694	269, 270, 271, 275, 276, 277, 282,	
<code>\dm</code>	61, 144	283, 284, 285, 286, 289, 291, 293,	
<code>\document</code>	14, 68	294, 296, 343, 344, 363, 364, 366,	
<code>drop-exponent</code>	22	368, 517, 836, 838, 911, 913, 914,	
<code>drop-uncertainty</code>	22	919, 920, 928, 930, 931, 936, 937,	
<code>drop-zero-decimal</code>	23	941, 943, 944, 1367, 1398, 1441, 1556	
<code>\dyne</code>	111, 960	<code>\exp_not:n</code>	38, 67, 108, 128, 163, 164,
		179, 182, 188, 191, 207, 245, 246,	
		253, 254, 255, 255, 260, 265, 267,	
		270, 275, 277, 287, 303, 313, 317,	
		353, 354, 362, 363, 433, 452, 455,	
		458, 461, 464, 467, 470, 472, 525,	
		553, 554, 555, 557, 587, 593, 619,	
		634, 635, 636, 637, 643, 646, 647,	
		647, 648, 648, 651, 654, 658, 659,	
		662, 666, 667, 671, 672, 680, 704,	
		724, 729, 729, 731, 732, 742, 743,	
		746, 766, 772, 774, 775, 777, 779,	
		780, 781, 782, 783, 785, 789, 790,	
		791, 791, 793, 794, 798, 799, 804,	
		807, 810, 813, 814, 815, 816, 830,	
		840, 852, 1118, 1121, 1125, 1157,	
		1164, 1166, 1199, 1203, 1206, 1208,	
		1244, 1246, 1249, 1250, 1336, 1337,	
		1361, 1362, 1367, 1376, 1378, 1394,	
		1400, 1402, 1412, 1415, 1434, 1443,	
		1444, 1457, 1458, 1462, 1466, 1483,	
		1488, 1490, 1496, 1497, 1498, 1499,	
		1545, 1553, 1558, 1559, 1561, 1573	
E			
<code>\electronvolt</code>	53, 54, 55, 56, 57, 58, 111, 904		
<code>\else</code>	143, 145		
else commands:			
<code>\else:</code>	338		
<code>\end</code>	54, 75, 190		
<code>\endinput</code>	13		
<code>\enquote</code>	73		
<code>\ensuremath</code>	24, 26, 32, 65, 106, 166, 244,		
	247, 287, 340, 364, 379, 383, 873,		
	908, 911, 919, 924, 925, 928, 936, 941		
<code>\erg</code>	111, 960		
<code>\ERROR</code>	56		
<code>\eV</code>	43, 145		
<code>evaluate-expression</code>	23		
<code>\exa</code>	109, 860		
exp commands:			
<code>\exp_after:wN</code> ...	56, 58, 107, 109,		
	111, 115, 125, 135, 137, 147, 167,		
	168, 198, 203, 211, 295, 337, 339,		
	355, 368, 369, 372, 444, 445, 446,		

\pA	2, 144		Q
\PackageError	7	\qty	58, 155, 163, 174
parse-numbers	24	\qtyInBookmark	157
parse-units	113	\qtylist	94, 168
\pascal	78, 79, 80, 81, 110, 883	\qtylistInBookmark	157
\pdfstringdefDisableCommands	154, 179	\qtyproduct	94, 170
peek commands:		\qtyproductInBookmark	157
\peek_after:Nw	343	\qtyrange	94, 172
\peek_catcode_ignore_spaces:NTF	40, 133, 328	\qtyrangeInBookmark	157
\per	103, 112, 113, 114, 118, 124, 124, 126, 127, 127, 129, 383, 391, 397, 923, 947, 986, 989	qualifier-mode	113
per-mode	113	qualifier-phrase	113
per-symbol	113	quantity-product	78
\percent	111, 970	quark commands:	
\peta	109, 860	\q_mark	239, 248, 517, 542, 568, 571, 624, 627, 636, 639, 644, 647, 649, 651, 658, 661
\pF	71, 146	\q_nil	46, 124, 195, 233, 254, 263, 371, 376, 467, 473, 516, 518, 542, 571, 627, 639, 647, 648, 651, 652, 661, 1427, 1429, 1431, 1451, 1511, 1538
\pg	35, 144	\quark_if_nil:NTF	1435, 1445, 1455, 1459, 1463, 1525
\phot	111, 960	\quark_if_nil:nTF	262
\pico	3, 16, 22, 36, 62, 73, 88, 109, 850	\quark_if_recursion_tail_stop:N	146, 198, 300, 305, 326, 537, 1623
\pm	61, 115, 144, 284, 333, 1489, 1662, 1663	\quark_if_recursion_tail_stop:n	213, 225
\pmol	14, 145	\quark_if_recursion_tail_stop_-do:Nn	297, 375, 397, 482, 499, 861, 870, 886, 900, 911, 932, 950, 980, 1054, 1083, 1097, 1129, 1137, 1612
\poise	111, 960	\q_recursion_stop	127, 192, 200, 204, 221, 292, 296, 314, 322, 371, 379, 483, 500, 541, 856, 864, 875, 879, 889, 914, 960, 1006, 1013, 1050, 1079, 1091, 1120, 1608, 1619
\power	112, 112	\q_recursion_tail	127, 192, 200, 204, 221, 292, 294, 321, 371, 856, 864, 875, 879, 889, 914, 960, 1006, 1013, 1050, 1079, 1091, 1120, 1607, 1619
prefix-mode	78	\q_stop	60, 61, 93, 125, 132, 178, 180, 195, 212, 214, 226, 233, 243, 247, 251, 255, 263, 268, 273, 356, 360, 373, 376, 469, 473, 521, 542, 549, 551, 563, 567, 568, 571, 579, 586, 624, 627, 636, 639, 644, 648, 649, 652, 658, 660, 661, 672, 677, 687, 695, 699, 705, 707, 717, 719, 774, 784, 788, 793
prg commands:			R
\prg_new_conditional:Npnn	1086, 1594	\radian	110, 883
\prg_new_protected_conditional:Npnn	11, 62, 1577	\raiseto	112, 115
\prg_replicate:nn	157, 304, 305, 317, 322, 619, 730, 819, 1109, 1247, 1517	\relax	55, 76
\prg_return_false:	19, 77, 1093, 1101, 1587, 1612		
\prg_return_true:	18, 73, 1098, 1591, 1629		
print-implicit-plus	24		
print-unity-mantissa	24		
print-zero-exponent	24		
prop commands:			
\prop_clear:N	299		
\prop_get:NnNTF	408, 545, 605, 610		
\prop_if_empty:NTF	151		
\prop_if_in:NnTF	313, 357, 403, 437		
\prop_if_in_p:Nn	337		
\prop_new:N	62, 63, 294		
\prop_put:Nnn	58, 59, 320, 421, 426		
\prop_remove:Nn	405, 417		
propagate-math-font	66		
\ProvideDocumentCommand	71		
\ProvidesExplPackage	15		
\ps	85, 144		
\pV	21, 145		

\siunitx_number_process:NN	20, 21, 45, 465, 511, 602
\siunitx_number_product:n	16, 59, 125	
\siunitx_number_range:nn	16, 105, 142	
\siunitx_print:nn	32,
	35, 37, 38, 48, 65, 65, 66, 66, 66,	
	67, 67, 78, 79, 81, 83, 86, 91, 108,	
	111, 200, 207, 214, 241, 264, 270,	
	476, 478, 552, 586, 600, 609, 676, 719	
\siunitx_print_match:n	65, 101
\siunitx_print_math:n	..	65, 104, 120
\l_siunitx_print_series_prop	...	67
\siunitx_print_text:n	
	65, 89, 102, 105, 112, 156, 249
\siunitx_quantity:nn	20, 51, 64, 78, 86	
\l_siunitx_unit_font_tl	
	108, 109, 112,
	121, 266, 276, 635, 647, 658, 671, 742	
\siunitx_unit_format:nN	
	29, 36, 47, 69, 78,
	81, 90, 106, 106, 107, 119, 129, 213, 269	
\siunitx_unit_format:nNN	40, 107, 129	
\siunitx_unit_options_apply:n	...	
	23, 62,
	79, 88, 88, 98, 108, 108, 115, 132, 304	
\siunitx_unit_pdfstring_context:	
	108, 284
\siunitx_unit_power_set:NnN	...	119
\l_siunitx_unit_seq	..	22, 75, 91, 108
\l_siunitx_unit_symbolic_seq	21,	
	24, 29, 29, 53, 66, 108, 108, 142, 287	
siunitx internal commands:		
__siunitx_angle:nnn	72, 187
__siunitx_angle_arc_convert:n	..	39
__siunitx_angle_arc_print:nnn	..	
	47, 128, 166
__siunitx_angle_arc_print_-		
auxi:nnn	166
__siunitx_angle_arc_print_-		
auxii:nw	178, 194
__siunitx_angle_arc_print_-		
auxiii:w	166
__siunitx_angle_arc_print_-		
auxiii:n	166
__siunitx_angle_arc_print_-		
auxiv:NN	166
__siunitx_angle_arc_print_-		
auxv:w	166
__siunitx_angle_arc_print_-		
auxvi:n	166
__siunitx_angle_arc_sign:nn	...	85
__siunitx_angle_arc_sign:nnn	60, 85	
\l_siunitx_angle_astronomy_bool	
	6, 177
\l__siunitx_angle_degrees_tl	...	
	44, 45, 46, 48, 81, 129
__siunitx_angle_extract_-		
sign:nnnnnnnn	85
\l__siunitx_angle_fill_degrees_-		
bool	6
\l__siunitx_angle_fill_minutes_-		
bool	6
\l__siunitx_angle_fill_seconds_-		
bool	6
\l__siunitx_angle_force_arc_bool	6, 41
\l__siunitx_angle_force_decimal_-		
bool	6, 55
\l__siunitx_angle_marker_box	...	
	164, 204, 218, 222, 228, 230, 234, 239	
\l__siunitx_angle_minutes_tl	81, 130	
\l__siunitx_angle_product_tl	6, 268	
\l__siunitx_angle_seconds_tl	81, 131	
\l__siunitx_angle_separator_tl	6, 186	
__siunitx_angle_sign:nnnnnnnn	...	85
\l__siunitx_angle_sign_tl	
	84, 95, 99, 108, 157
\l__siunitx_angle_tmp_bool	
	3, 198, 199, 246
\l__siunitx_angle_tmp_dim	
	3, 231, 253, 255
\l__siunitx_angle_tmp_tl	
	..	3, 146, 147, 153, 213, 214, 269, 270
\l__siunitx_angle_unit_box	
	164, 209, 219, 223, 227, 236
__siunitx_bookmark_cmd:Nn	...	157
__siunitx_bookmark_cmd:Nnn	...	
	157, 163, 164,
	165, 166, 167, 168, 169, 170, 171, 172	
\c__siunitx_bookmark_seq	...	173, 181
__siunitx_command_create:	37
__siunitx_command_create:N	...	37
\l__siunitx_command_create_bool	..	
	4, 39
\l__siunitx_command_optarg_bool	..	
	4, 71, 75
\l__siunitx_command_overwrite_-		
bool	4, 64
\l__siunitx_command_prespace_-		
bool	4, 80
\l__siunitx_command_tmp_tl	..	3, 82, 84
\l__siunitx_command_xspace_bool	..	
	4, 62, 88
__siunitx_declare_column:Nnn	...	198
\l__siunitx_list_bracket_bool	...	6
\l__siunitx_list_repeat_bool	...	6
\l__siunitx_list_separator_-		
final_tl	6, 43

\l_siunitx_list_separator_pair_-	_siunitx_number_exponent_-
tl 6, 41	finalise:n 626
\l_siunitx_list_separator_tl . 6, 42	_siunitx_number_exponent_-
\l_siunitx_list_tmp_seq 3, 33, 37, 40	finalise:n 667, 681, 737, 767, 778, 786, 1069
\l_siunitx_list_tmp_tl 3, 36, 38	_siunitx_number_exponent_-
_siunitx_load_check: 17	fixed:nnnnnn 626
_siunitx_load_check:n ... 20, 32, 36	\l_siunitx_number_exponent_-
\l_siunitx_number_arg_tl 32, 35,	fixed_int 567, 645, 648
68, 86, 92, 94, 212, 219, 222, 238,	_siunitx_number_exponent_-
253, 265, 327, 343, 370, 549, 555, 563	input:nnnnnn 626
\l_siunitx_number_bracket_-	\l_siunitx_number_exponent_-
close_tl 1253, 1378	mode_tl 567, 631, 635, 1029
\l_siunitx_number_bracket_-	\l_siunitx_number_exponent_-
negative_bool 1253, 1351	product_tl 1253, 1558
\l_siunitx_number_bracket_open_-	_siunitx_number_exponent_-
tl 1253, 1376	scientific:nnnnnn 626
\l_siunitx_number_comparator_tl	_siunitx_number_exponent_-
..... 69, 218, 221, 353	scientific:nnnw 626
_siunitx_number_digits:NN 610, 797	_siunitx_number_exponent_-
_siunitx_number_digits:Nn ... 797	shift:nnn 626, 1063
_siunitx_number_digits:nnnnnn 797	_siunitx_number_exponent_-
_siunitx_number_drop_exponent:NN	shift_down:nnn 626
..... 608, 821	_siunitx_number_exponent_-
_siunitx_number_drop_exponent:nnnnnn	shift_down:nnnw 626
..... 821	_siunitx_number_exponent_-
\l_siunitx_number_drop_exponent_-	shift_down:nw 626
bool 567, 823	_siunitx_number_exponent_-
_siunitx_number_drop_uncertainty:NN	shift_up:nnn 626
..... 606, 831	_siunitx_number_exponent_-
_siunitx_number_drop_uncertainty:nnnnnn	shift_up:nnw 626
..... 831	_siunitx_number_exponent_-
\l_siunitx_number_drop_uncertainty_-	shift_up_aux:nnn 626
bool 567, 833	\l_siunitx_number_exponent_tl ..
\l_siunitx_number_drop_zero_- 70,
decimal_bool 567, 1236	229, 263, 278, 286, 287, 298, 308, 356
\l_siunitx_number_explicit_-	_siunitx_number_expression:n ..
plus_bool 30, 284, 558 30, 89
_siunitx_number_exponent:NN ...	\l_siunitx_number_expression_-
..... 622, 626	bool 30, 88
\l_siunitx_number_exponent_-	\l_siunitx_number_flex_tl 29, 71,
base_tl 1253, 1561	102, 111, 115, 137, 145, 462, 464, 543
_siunitx_number_exponent_-	_siunitx_number_format:Nn .. 1314
engineering:nnnnnn 626	_siunitx_number_format:nN 14
_siunitx_number_exponent_-	_siunitx_number_format:nnnnnn
engineering:nNw 626 1314
_siunitx_number_exponent_-	_siunitx_number_format_-
engineering_0:nnnn 626	comparator:nn 1314
_siunitx_number_exponent_-	_siunitx_number_format_-
engineering_1:nnnn 626	decimal:nn 1314
_siunitx_number_exponent_-	_siunitx_number_format_-
engineering_2:nnnn 626	decimal_aux:n 1314
_siunitx_number_exponent_-	_siunitx_number_format_-
engineering_aux:nnnnnn 626	decimal_loop:NNNN 1314

_siunitx_number_format_digits:nn	1314	_siunitx_number_if_number_token_auxiii:NN	1594
_siunitx_number_format_end:	1314	\l_siunitx_number_implicit_plus_bool	1253, 1343, 1566
_siunitx_number_format_exponent:nnnn	1314	\l_siunitx_number_input_comparator_tl	30, 216, 1600
_siunitx_number_format_integer:nnn	1314	\l_siunitx_number_input_digit_tl	30, 306, 381, 399, 484, 501, 1601
_siunitx_number_format_integer_aux:n	1314	\l_siunitx_number_input_exponent_tl	30, 227, 235, 236, 1602
_siunitx_number_format_integer_aux_0:n	1314	\l_siunitx_number_input_ignore_tl	30, 1603
_siunitx_number_format_integer_aux_1:n	1314	\l_siunitx_number_input_sign_tl	30, 275, 414, 553, 1605
_siunitx_number_format_integer_aux_2:n	1314	\l_siunitx_number_input_tl	73, 92, 128
_siunitx_number_format_integer_first:nnNN	1314	\l_siunitx_number_input_uncert_close_tl	30, 516, 1599
_siunitx_number_format_integer_loop:NNNN	1314	\l_siunitx_number_input_uncert_open_tl	30, 409, 1604
_siunitx_number_format_sign:N	1314	\l_siunitx_number_input_uncert_sign_tl	30, 116, 1606
_siunitx_number_format_sign:n	1314	\l_siunitx_number_min_decimal_int	567, 811
_siunitx_number_format_sign_brackets:w	1314	\l_siunitx_number_min_integer_int	567, 806
_siunitx_number_format_sign_color:w	1314	\l_siunitx_number_negative_color_tl	1253, 1349, 1367
_siunitx_number_format_uncertainty:nnn	1314	_siunitx_number_number_token_auxi:NN	1597, 1610, 1614
_siunitx_number_format_uncertainty_S:nnnw	1314	_siunitx_number_number_token_auxii:NN	1613, 1616
_siunitx_number_format_uncertainty_S:nnw	1314	_siunitx_number_number_token_auxiii:NN	1618, 1621, 1632
_siunitx_number_format_uncertainty_S_aux:nnn	1314	\l_siunitx_number_output_uncert_close_tl	1253, 1499
_siunitx_number_format_uncertainty_S_aux:nnnw	1507, 1523, 1530, 1537	\l_siunitx_number_output_uncert_open_tl	1253, 1497
_siunitx_number_format_uncertainty_S_aux:nnw	1528, 1538	_siunitx_number_parse:nN	76
_siunitx_number_format_uncertainty_unaligned:n	1314	_siunitx_number_parse_check:	96, 100
\l_siunitx_number_formatted_tl	8, 21, 24, 26	_siunitx_number_parse_combine_uncert:	118, 133
\l_siunitx_number_group_decimal_bool	1253	_siunitx_number_parse_combine_uncert_auxi:nnnnnnnn	133
\l_siunitx_number_group_integer_bool	1253	_siunitx_number_parse_combine_uncert_auxii:nnnnnn	133
\l_siunitx_number_group_minimum_int	1253, 1411	_siunitx_number_parse_combine_uncert_auxiii:nnnnnn	133
\l_siunitx_number_group_separator_tl	1253, 1440, 1443, 1468	_siunitx_number_parse_combine_uncert_auxiv:nnnn	133
_siunitx_number_if_number_token_auxi:NN	1594	_siunitx_number_parse_combine_uncert_auxv:w	133
_siunitx_number_if_number_token_auxii:NN	1594		

_siunitx_number_parse_combine- uncert_auxvi:w	133	_siunitx_number_parse_loop- root_swap:NNwNN	365
_siunitx_number_parse_comparator:	95, 209	_siunitx_number_parse_loop- uncert:NNNNN	365
_siunitx_number_parse_comparator- aux:Nw	209	_siunitx_number_parse_replace:	93, 316
_siunitx_number_parse_exponent:	225, 565	_siunitx_number_parse_replace- aux:nN	316
_siunitx_number_parse_exponent- auxi:w	225	_siunitx_number_parse_replace- minus:	318, 341
_siunitx_number_parse_exponent- auxii:nn	225	_siunitx_number_parse_replace- sign:	316
_siunitx_number_parse_exponent- auxiii:Nw	225	_siunitx_number_parse_sign:	223, 546
_siunitx_number_parse_exponent- auxiv:nn	225	_siunitx_number_parse_sign- aux:Nw	546
_siunitx_number_parse_exponent- check:N	225	\c_siunitx_number_parse_sign- replacement_tl	316
_siunitx_number_parse_exponent- cleanup:N	225	\l_siunitx_number_parsed_tl	19, 20, 22, 30, 72, 85, 98, 107, 119, 121, 123, 137, 144, 179, 181, 230, 267, 270, 279, 289, 315, 349, 351, 354, 369, 544, 559, 560, 562, 564, 1583, 1584
_siunitx_number_parse_exponent- cleanup:wN	311, 313	\l_siunitx_number_partial_tl 74, 367, 429, 430, 433, 443, 468, 469, 472, 476, 486, 506, 521, 524, 525	
_siunitx_number_parse_exponent- zero_test:N	225	_siunitx_number_process:nnnnnnn	602
_siunitx_number_parse_finalise:	131, 347	_siunitx_number_round:NN	623, 841
_siunitx_number_parse_finalise:nw	347	_siunitx_number_round:nnn	853, 1141, 1177, 1187, 1215, 1224, 1229
_siunitx_number_parse_loop:	231, 271, 365	_siunitx_number_round_auxi:nnnN	853
_siunitx_number_parse_loop- after_decimal:NNN	365	_siunitx_number_round_auxii:nnnN	853
_siunitx_number_parse_loop- after_uncert:N	365	_siunitx_number_round_auxiii:nnnN	853
_siunitx_number_parse_loop- break:wN	365	_siunitx_number_round_auxiv:nnN	853
_siunitx_number_parse_loop- first:N	365	_siunitx_number_round_auxiv:nnnN	874, 888, 898, 904
_siunitx_number_parse_loop- first:NNN	368, 373, 463	_siunitx_number_round_auxv:nnN	853
_siunitx_number_parse_loop- main:NNNNN	365	_siunitx_number_round_auxvi:nN	853
_siunitx_number_parse_loop- main_decimal:NN	365	_siunitx_number_round_auxvi:nnN	913
_siunitx_number_parse_loop- main_digit:NNNNN	365	_siunitx_number_round_auxvi:nnnN	907, 930
_siunitx_number_parse_loop- main_end:NN	365	_siunitx_number_round_auxvii:nnN	853
_siunitx_number_parse_loop- main_sign:NNN	365	_siunitx_number_round_auxviii:nnN	853
_siunitx_number_parse_loop- main_store:NNN	365	_siunitx_number_round_engineering:nn	853
_siunitx_number_parse_loop- main_uncert:NNN	365		

_siunitx_number_round_engineering:nnN	_siunitx_number_round_scientific:nn
..... 853 853
_siunitx_number_round_engineering:NNNNn	_siunitx_number_round_truncate:n
..... 853 853
_siunitx_number_round_figures:nnnnnn	_siunitx_number_round_truncate:nnN
..... 1112 853
_siunitx_number_round_figures_-	_siunitx_number_round_truncate_-
count:nnN 1112	direct:n 853, 1231
_siunitx_number_round_figures_-	_siunitx_number_round_uncertainty:nnn
count:nnnN 1112 1194
_siunitx_number_round_final:nn	_siunitx_number_round_uncertainty:nnnnn
..... 853 1194
_siunitx_number_round_final_-	_siunitx_number_round_uncertainty:nnnnnnn
decimal:nnw 853 1194
_siunitx_number_round_final_- 1194
integer:nnw 853	\l_siunitx_number_tight_bool ...
_siunitx_number_round_final_- 1253, 1360, 1555
output:nn 853	\l_siunitx_number_tmp_tl 7,
_siunitx_number_round_final_-	114, 117, 234, 239, 245, 246, 254, 255
shift:nn 853	\l_siunitx_number_uncert_-
_siunitx_number_round_final_-	separate_bool 1253, 1486
shift:Nw 853	\l_siunitx_number_uncert_-
_siunitx_number_round_fixed:nn	separator_tl 1253, 1496
..... 853	\l_siunitx_number_unity_-
\l_siunitx_number_round_half_-	mantissa_bool ... 1253, 1388, 1552
even_bool 567, 921, 939	\l_siunitx_number_valid_tl .. 1576
_siunitx_number_round_if_-	\l_siunitx_number_validate_bool
half:N 1086 75, 125, 1581
_siunitx_number_round_if_-	_siunitx_number_zero_decimal:NN
half:n 1086 609, 1234
_siunitx_number_round_if_half_-	_siunitx_number_zero_decimal:nnnnnnn
p:n 923, 941, 1086 1234
_siunitx_number_round_input:nn	\l_siunitx_number_zero_exponent_-
..... 853	bool 1253, 1547
\l_siunitx_number_round_min_tl	_siunitx_option_deprecated:nn .
..... 567, 846, 1026, 1072	10, 85, 91, 261, 267, 287, 338, 344,
_siunitx_number_round_none:nnnnnnn	350, 356, 519, 525, 537, 543, 549, 564
..... 841	_siunitx_option_deprecated:nnn
_siunitx_number_round_pad:nnn 10, 58,
..... 1103, 1146, 1172	66, 74, 100, 278, 298, 306, 313, 321,
\l_siunitx_number_round_pad_-	329, 362, 373, 381, 398, 492, 500, 512
bool 567, 1108	_siunitx_option_table_comparator:nnnnnnn
_siunitx_number_round_places:nnnnnnn 436
..... 1153	_siunitx_option_table_comparator:nnnnnnnn
_siunitx_number_round_places_- 451
decimal:nn 1153	_siunitx_option_table_figures-decimal:nnnnnnnn
_siunitx_number_round_places_- 436
end:nn 1068, 1153	_siunitx_option_table_figures-exponent:nnnnnnnn
_siunitx_number_round_places_- 436
integer:nn 1153	_siunitx_option_table_figures-integer:nnnnnnnn
\l_siunitx_number_round_- 436
precision_int 567,	_siunitx_option_table_figures-uncertainty:nnnnnnnn
1049, 1116, 1139, 1142, 1147, 1158, 436
1170, 1173, 1180, 1190, 1198, 1216	_siunitx_option_table_format:n
_siunitx_number_round_scientific:nn	436, 474, 476, 478, 480, 482, 484, 486
..... 1061	

_siunitx_option_table_sign-exponent:nnnn	_siunitx_print_text_scripts_-
..... 436	two:NnNn 249
_siunitx_option_table_sign-mantissa:nnnn	_siunitx_print_text_series_-
..... 436	bool 57, 258
_siunitx_print_convert_-	\l_siunitx_print_text_series_tl 34
series:n 120	\l_siunitx_print_text_shape_-
_siunitx_print_extract_-	bool 59, 263
series:Nw 120	\l_siunitx_print_text_shape_tl . 34
_siunitx_print_math_aux:Nn .. 120	_siunitx_print_text_sub:n ... 249
_siunitx_print_math_auxi:n .. 120	_siunitx_print_text_super:n .. 249
_siunitx_print_math_auxii:n .. 120	\l_siunitx_print_tmp_box ... 12, 24
_siunitx_print_math_auxiii:n .. 120	\l_siunitx_print_tmp_tl 12, 124,
_siunitx_print_math_auxiv:n .. 120	126, 128, 208, 209, 210, 213, 216,
_siunitx_print_math_auxv:n .. 120	230, 231, 232, 241, 242, 244, 274,
\l_siunitx_print_math_family_-	275, 276, 313, 314, 315, 349, 350, 352
bool 34, 169	\l_siunitx_print_unit_color_tl . 34
\l_siunitx_print_math_font_bool	\l_siunitx_print_unit_mode_tl .. 34
..... 34, 184	\l_siunitx_print_version_b_tl .. 73
_siunitx_print_math_script:n . 120	\l_siunitx_print_version_eb_tl . 73
_siunitx_print_math_sub:n ... 120	\l_siunitx_print_version_el_tl . 73
_siunitx_print_math_super:n .. 120	\l_siunitx_print_version_l_tl .. 73
_siunitx_print_math_text:n .. 120	\l_siunitx_print_version_m_tl .. 73
_siunitx_print_math_version:nn 120	\l_siunitx_print_version_sb_tl . 73
\l_siunitx_print_math_version_-	\l_siunitx_print_version_sl_tl . 73
bool 34, 146	\l_siunitx_print_version_ub_tl . 73
\l_siunitx_print_math_weight_-	\l_siunitx_print_version_ul_tl . 73
bool 34, 122	\c_siunitx_print_weight_b_tl .. 118
\c_siunitx_print_mathrm_int 14, 200	\c_siunitx_print_weight_c_tl .. 116
\c_siunitx_print_mathsf_int 14, 188	\c_siunitx_print_weight_ecl_tl 116
\c_siunitx_print_mathtt_int 14, 189	\c_siunitx_print_weight_ex_tl . 116
\l_siunitx_print_number_color_-	\c_siunitx_print_weight_l_tl .. 118
tl 34	\c_siunitx_print_weight_m_tl .. 118
\l_siunitx_print_number_mode_tl 34	\c_siunitx_print_weight_sc_tl . 116
_siunitx_print_replace_font:N .	\c_siunitx_print_weight_sx_tl . 116
..... 107, 209, 231, 281	\c_siunitx_print_weight_uc_tl . 116
_siunitx_print_store_fam:n ... 14	\c_siunitx_print_weight_ux_tl . 116
\l_siunitx_print_text_family_-	\c_siunitx_print_weight_x_tl .. 116
bool 55, 253	\l_siunitx_product_mode_tl . 49, 70
\l_siunitx_print_text_family_tl 34	_siunitx_product_phrase: 59
_siunitx_print_text_replace:N 249	\l_siunitx_product_phrase_tl 49, 74
_siunitx_print_text_replace:n 249	_siunitx_product_symbol: 59
_siunitx_print_text_replace:NNn	\l_siunitx_product_symbol_tl 49, 78
..... 249	\l_siunitx_product_tmp_seq
_siunitx_print_text_scripts: . 249 47, 62, 66, 69
_siunitx_print_text_scripts:NnN	\l_siunitx_product_tmp_tl 47, 65, 67, 77, 79
..... 249	_siunitx_quantity_adjust_exp:n 49
_siunitx_print_text_scripts_-	_siunitx_quantity_adjust_-
one:Nn 339, 346, 377	exp:nNN 43, 49
_siunitx_print_text_scripts_-	_siunitx_quantity_adjust_-
one:NnN 249	exp:nnnnnnnNN 49
_siunitx_print_text_scripts_-	_siunitx_quantity_adjust_-
two:n 249	exp:Nw 49
_siunitx_print_text_scripts_-	
two:nn 249	

`\l__siunitx_quantity_break_bool` .
 [5](#), [34](#)
`__siunitx_quantity_combine-`
 prefix:nn [31](#), [38](#)
`\l__siunitx_quantity_number_tl` ..
 [18](#), [28](#), [32](#), [46](#)
`\l__siunitx_quantity_prefix_bool`
 [5](#), [25](#)
`\l__siunitx_quantity_product_tl` .
 [5](#), [33](#)
`\l__siunitx_quantity_tmp_fp` [3](#), [41](#), [44](#)
`\l__siunitx_quantity_tmp_tl`
 [3](#), [42](#), [44](#), [45](#), [47](#)
`\l__siunitx_quantity_unit_tl`
 [18](#), [29](#), [35](#), [41](#)
`\l__siunitx_range_bracket_bool` .. [83](#)
`\l__siunitx_range_phrase_tl` . [83](#), [109](#)
`\l__siunitx_range_repeat_bool` ... [83](#)
`\l__siunitx_range_tmp_tl`
 [82](#), [107](#), [108](#), [110](#), [111](#)
`__siunitx_symbol_if_replace:Nn` . [62](#)
`__siunitx_symbol_if_replace:NnTF`
 [62](#), [82](#), [94](#), [108](#)
`__siunitx_symbol_mathOmega:` ...
 [24](#), [105](#), [159](#)
`__siunitx_symbol_non_latin:n` [41](#),
 [65](#), [90](#), [103](#), [117](#), [158](#), [160](#), [162](#), [164](#)
`__siunitx_symbol_non_latin:nnnn` [41](#)
`__siunitx_symbol_textmu:` [13](#), [118](#), [157](#)
`__siunitx_symbol_texttimes:` .. [123](#)
`\l__siunitx_symbol_tmpa_tl`
 .. [3](#), [65](#), [66](#), [67](#), [70](#), [132](#), [134](#), [135](#), [137](#)
`\l__siunitx_symbol_tmpb_tl`
 [3](#), [69](#), [70](#), [136](#), [137](#)
`\l__siunitx_table_after_box` [416](#),
 [443](#), [445](#), [449](#), [456](#), [459](#), [470](#), [523](#), [536](#)
`\l__siunitx_table_after_model_tl`
 [271](#), [284](#), [426](#), [522](#)
`\l__siunitx_table_after_tl`
 [101](#), [112](#), [119](#), [147](#)
`\l__siunitx_table_align_after-`
 bool [419](#), [526](#)
`__siunitx_table_align_auxi:nn` . [183](#)
`__siunitx_table_align_auxii:nn` [183](#)
`\l__siunitx_table_align_before-`
 bool [419](#), [545](#), [590](#)
`__siunitx_table_align_center:n` [183](#)
`\l__siunitx_table_align_comparator-`
 bool [419](#), [575](#)
`\l__siunitx_table_align_exponent-`
 bool [419](#), [668](#)
`__siunitx_table_align_left:n` .. [183](#)
`\l__siunitx_table_align_mode_tl` .
 [257](#), [332](#), [336](#), [433](#)
`\l__siunitx_table_align_number-`
 tl [257](#), [408](#), [531](#), [715](#)
`__siunitx_table_align_right:n` . [183](#)
`\l__siunitx_table_align_text_tl` .
 [216](#), [226](#)
`\l__siunitx_table_align_uncertainty-`
 bool [419](#), [657](#)
`\l__siunitx_table_auto_round-`
 bool [257](#), [500](#)
`\l__siunitx_table_before_box` ...
 [416](#), [437](#), [438](#), [440](#), [446](#), [448](#),
 [452](#), [457](#), [463](#), [486](#), [487](#), [489](#), [492](#),
 [496](#), [533](#), [557](#), [559](#), [565](#), [611](#), [613](#), [620](#)
`\l__siunitx_table_before_model-`
 tl [269](#), [282](#), [433](#), [485](#)
`\l__siunitx_table_before_tl`
 [101](#), [110](#), [114](#), [117](#)
`\l__siunitx_table_carry_dim`
 [418](#), [524](#), [527](#), [631](#), [686](#), [694](#), [706](#)
`__siunitx_table_center_marker:` .
 [235](#), [364](#), [479](#)
`__siunitx_table_cleanup-`
 decimal:w [232](#), [379](#), [478](#)
`__siunitx_table_collect_begin:` .
 [11](#), [24](#)
`__siunitx_table_collect_begin:N` [96](#)
`__siunitx_table_collect_begin:w` [24](#)
`__siunitx_table_collect_end:` [19](#), [104](#)
`__siunitx_table_collect_group:n` [38](#)
`__siunitx_table_collect_loop:` ..
 [30](#), [37](#), [38](#)
`__siunitx_table_collect-`
 search:NnTF [38](#)
`__siunitx_table_collect_search-`
 aux:Nn [38](#)
`\l__siunitx_table_collect_tl` ...
 [23](#), [26](#), [46](#), [60](#), [81](#), [106](#), [107](#), [109](#)
`__siunitx_table_collect_token:N` [38](#)
`\l__siunitx_table_column_width-`
 dim [176](#), [192](#)
`\l__siunitx_table_decimal_box` ...
 . [229](#), [239](#), [241](#), [244](#), [250](#), [341](#), [356](#),
 [366](#), [380](#), [398](#), [399](#), [411](#), [477](#), [481](#),
 [535](#), [630](#), [634](#), [683](#), [685](#), [690](#), [701](#), [703](#)
`__siunitx_table_direct_begin:` ..
 [12](#), [323](#)
`__siunitx_table_direct_begin:w` [323](#)
`__siunitx_table_direct_end:` [20](#), [323](#)
`__siunitx_table_direct_format:` [323](#)
`__siunitx_table_direct_format:nnnnnn`
 [323](#)
`__siunitx_table_direct_format:w` [323](#)
`__siunitx_table_direct_format-`
 aux:w [372](#), [375](#)

_siunitx_table_direct_format_- end:	323	_siunitx_table_print_format_- auxvii:w	649, 658, 660
_siunitx_table_direct_format_- switch:	323	_siunitx_table_print_format_- box:Nn	432
_siunitx_table_direct_marker:	323	_siunitx_table_print_marker:nnn	432
_siunitx_table_direct_marker_- end:	323	_siunitx_table_print_marker:w	432
_siunitx_table_direct_marker_- switch:	323	_siunitx_table_print_marker_- auxi:w	432
_siunitx_table_direct_none:	323	_siunitx_table_print_marker_- auxii:w	432
_siunitx_table_direct_none_- end:	323	_siunitx_table_print_marker_- auxiii:w	432
_siunitx_table_fil:	231, 245, 252, 343, 382, 392, 406, 451, 460, 495, 529, 550, 564, 585, 598, 619, 691	_siunitx_table_print_none:nnn	432
\l_siunitx_table_fixed_width_- bool	176, 191	_siunitx_table_print_text:n	114, 223, 329
\l_siunitx_table_format_tl	281, 290, 292, 293, 296, 441, 445, 449, 508	_siunitx_table_skip:n	171, 195, 197, 209, 211
_siunitx_table_generate_- model:n	272, 285	_siunitx_table_split:nNNN	108, 122, 268
_siunitx_table_generate_- model:nnnnnn	285, 448	_siunitx_table_split_group:NNNn	122
_siunitx_table_generate_model_- S:nw	285	_siunitx_table_split_loop:NNN	122
\l_siunitx_table_integer_box	229, 238, 242, 249, 253, 344, 365, 384, 410, 475, 480, 534, 547, 556, 561, 576, 578, 580, 584, 592, 594, 599, 605, 606, 608, 616	_siunitx_table_split_tidy:N	128, 129, 160
\l_siunitx_table_model_tl	270, 272, 281, 301, 371, 516	_siunitx_table_split_tidy:Nn	160
\l_siunitx_table_number_tl	101, 111, 113, 118	_siunitx_table_split_token:NNNN	122
_siunitx_table_print:nnn	116, 432	\l_siunitx_table_text_bool	6, 9, 16, 225
_siunitx_table_print_format:nnn	432	\l_siunitx_table_tmp_box	3, 339, 342, 378, 381, 383, 385, 485, 493, 522, 524, 544, 548, 560, 573, 581, 595, 614, 629, 633, 654, 655, 656, 665, 666, 667, 687, 692, 697, 704, 709
_siunitx_table_print_format:nnnnn	507, 539	\l_siunitx_table_tmp_dim	3, 101, 604, 615, 655, 666, 696, 708
_siunitx_table_print_format_- after:N	432	\l_siunitx_table_tmp_tl	3, 370, 373, 464, 465, 466, 467, 469, 498, 511, 513, 514, 518, 521, 718, 719
_siunitx_table_print_format_- auxi:w	520, 541	_siunitx_tmp:w 23, 24, 28, 29, 201, 203	
_siunitx_table_print_format_- auxii:w	568, 570	\l_siunitx_tmp_tl	36, 37, 38, 47, 48, 80, 81, 90, 91
_siunitx_table_print_format_- auxiii:w	624, 626	\l_siunitx_unit_autofrac_bool	454, 461, 468, 475, 482, 489, 508, 761
_siunitx_table_print_format_- auxiv:w	636, 638	\l_siunitx_unit_bracket_bool	506, 530, 551, 595, 685, 706
_siunitx_table_print_format_- auxv:w	642, 646	\l_siunitx_unit_bracket_close_- tl	441, 555, 637
_siunitx_table_print_format_- auxvi:w	643, 650	\l_siunitx_unit_bracket_open_tl	441, 553, 634
		\l_siunitx_unit_current_tl	516, 531, 590, 592, 619, 627, 665, 672, 680, 732, 740, 743

<code>\l_siunitx_unit_denominator_-</code>	<code>_siunitx_unit_format_parsed: ..</code>
<code>bracket_bool</code> 441 , 704 152 , 520
<code>\l_siunitx_unit_denominator_tl</code> .	<code>_siunitx_unit_format_parsed_-</code>
..... 518 , 523 , 705 , 750 , 791 , 800 , 809 , 816	<code>aux:n</code> 520
<code>\l_siunitx_unit_font_bool</code>	<code>_siunitx_unit_format_power: ..</code> 559
..... 507 , 532 , 669 , 675 , 738 , 745	<code>_siunitx_unit_format_power_-</code>
<code>\l_siunitx_unit_forbid_literal_-</code>	<code>aux:wTF</code> 559
<code>bool</code> 155 , 441	<code>_siunitx_unit_format_power_-</code>
<code>_siunitx_unit_format:nNN</code> 129	<code>negative:</code> 559
<code>_siunitx_unit_format_aux:</code> ... 129	<code>_siunitx_unit_format_power_-</code>
<code>_siunitx_unit_format_bracket:N</code>	<code>negative_aux:w</code> 559
..... 549 , 592 , 800	<code>_siunitx_unit_format_power_-</code>
<code>_siunitx_unit_format_finalise:</code>	<code>positive:</code> 559
..... 539 , 748	<code>_siunitx_unit_format_power_-</code>
<code>_siunitx_unit_format_finalise_-</code>	<code>superscript:</code> 559
<code>autofrac:</code> 748	<code>_siunitx_unit_format_prefix: .</code> 597
<code>_siunitx_unit_format_finalise_-</code>	<code>_siunitx_unit_format_prefix_-</code>
<code>fraction:</code> 766 , 772 , 785	<code>power:</code> 597
<code>_siunitx_unit_format_finalise_-</code>	<code>_siunitx_unit_format_prefix_-</code>
<code>fractional:</code> 748	<code>symbol:</code> 597
<code>_siunitx_unit_format_finalise_-</code>	<code>_siunitx_unit_format_qualifier:</code>
<code>power:</code> 748 620
<code>_siunitx_unit_format_finalise_-</code>	<code>_siunitx_unit_format_qualifier_-</code>
<code>symbol:</code> 765 , 775 , 794	<code>bracket:</code> 620
<code>_siunitx_unit_format_font:</code> ...	<code>_siunitx_unit_format_qualifier_-</code>
..... 561 , 631 , 643 , 653 , 684 , 736	<code>combine:</code> 620
<code>_siunitx_unit_format_literal:n</code>	<code>_siunitx_unit_format_qualifier_-</code>
..... 157 , 160 , 174	<code>phrase:</code> 620
<code>_siunitx_unit_format_literal_-</code>	<code>_siunitx_unit_format_qualifier_-</code>
<code>auxi:w</code> 174	<code>subscript:</code> 620
<code>_siunitx_unit_format_literal_-</code>	<code>_siunitx_unit_format_special:</code> 663
<code>auxii:n</code> 214 , 218	<code>_siunitx_unit_format_unit: ..</code> 677
<code>_siunitx_unit_format_literal_-</code>	<code>\l_siunitx_unit_formatted_tl</code> ...
<code>auxii:w</code> 174 128 , 130 , 144 , 164 , 195 , 203 ,
<code>_siunitx_unit_format_literal_-</code>	204 , 253 , 256 , 258 , 524 , 713 , 714 ,
<code>auxiii:w</code> 174	759 , 760 , 774 , 776 , 780 , 781 , 782 ,
<code>_siunitx_unit_format_literal_-</code>	787 , 790 , 796 , 798 , 805 , 808 , 812 , 814
<code>auxiv:w</code> 174	<code>\l_siunitx_unit_fraction_-</code>
<code>_siunitx_unit_format_literal_-</code>	<code>function_tl</code> 441 , 789
<code>auxv:w</code> 174	<code>_siunitx_unit_if_symbolic:n</code> ... 11
<code>_siunitx_unit_format_literal_-</code>	<code>_siunitx_unit_if_symbolic:nTF</code> .
<code>subscript:</code> 174 11 , 79 , 148
<code>_siunitx_unit_format_literal_-</code>	<code>_siunitx_unit_literal_power:nn</code>
<code>superscript:</code> 174 43 , 116 , 172
<code>_siunitx_unit_format_literal_-</code>	<code>_siunitx_unit_literal_special:nN</code>
<code>tilde:</code> 174 107 , 110 , 173
<code>_siunitx_unit_format_output: ..</code>	<code>\c_siunitx_unit_math_subscript_-</code>
..... 537 , 682	<code>tl</code> 7 , 184 , 208 ,
<code>_siunitx_unit_format_output_-</code>	237 , 238 , 241 , 242 , 246 , 247 , 249 ,
<code>aux:</code> 682	250 , 273 , 656 , 914 , 920 , 931 , 937 , 944
<code>_siunitx_unit_format_output_-</code>	<code>_siunitx_unit_non_latin:n</code>
<code>aux:nn</code> 682 820 , 856 , 884 , 951
<code>_siunitx_unit_format_output_-</code>	<code>_siunitx_unit_non_latin:nnnn</code> . 820
<code>denominator:</code> 682	

\l__siunitx_unit_numerator_bool .	\l__siunitx_unit_product_tl
..... 134 , 512 , 533 , 573 , 689 121 , 215 , 697 , 708 , 815
\l__siunitx_unit_options_bool . . .	\l__siunitx_unit_qualifier_mode_-
..... 87 , 90 , 101	tl 441 , 513 , 625
__siunitx_unit_parse:n 150 , 297	\l__siunitx_unit_qualifier_-
__siunitx_unit_parse_add:nnnn . .	phrase_tl 441 , 646
310 , 327 , 341 , 359 , 376 , 381 , 395	\l__siunitx_unit_separator_tl . . 174
\l__siunitx_unit_parse_bool 146 , 441	__siunitx_unit_set_symbolic:Nnn
__siunitx_unit_parse_finalise: 23 , 42 , 45 , 51 , 66 , 76 , 103
..... 308 , 431	__siunitx_unit_set_symbolic:Nnnn
__siunitx_unit_parse_finalise:n 23
..... 307 , 400	__siunitx_unit_set_symbolic:Npnn
__siunitx_unit_parse_per: . 105 , 386 23 , 106 , 109 , 112 , 115 , 118
__siunitx_unit_parse_power:nnN .	\l__siunitx_unit_sticky_per_bool
..... 44 , 47 , 117 , 120 , 324 126 , 290 , 388
__siunitx_unit_parse_prefix:Nn .	\l__siunitx_unit_test_bool
..... 53 , 324 10 , 14 , 32 , 302
__siunitx_unit_parse_qualifier:nn	\l__siunitx_unit_tmp_fp 4 , 132
..... 68 , 114 , 324	\l__siunitx_unit_tmp_int . 4 , 326 , 328
__siunitx_unit_parse_special:n .	\l__siunitx_unit_tmp_tl
..... 108 , 111 , 324 4 , 15 , 17 , 180 , 181 , 183 ,
__siunitx_unit_parse_unit:Nn 81 , 373	193 , 194 , 196 , 199 , 312 , 314 , 321 ,
\l__siunitx_unit_parsed_prop . . .	332 , 338 , 355 , 357 , 402 , 403 , 406 ,
..... 124 , 151 ,	407 , 410 , 418 , 422 , 427 , 435 , 437 ,
294 , 299 , 313 , 320 , 338 , 357 , 403 ,	543 , 546 , 608 , 611 , 612 , 614 , 774 , 779
405 , 409 , 417 , 421 , 426 , 437 , 545 , 610	\l__siunitx_unit_total_int
\l__siunitx_unit_parsing_bool 519 , 522 , 528
..... 9 , 34 , 179 , 286 , 300	\l__siunitx_unit_two_part_bool . .
\l__siunitx_unit_part_tl 411 , 413 ,	457 , 464 , 471 , 478 , 485 , 492 , 508 , 701
414 , 415 , 422 , 516 , 546 , 563 , 576 ,	skip commands:
579 , 581 , 593 , 606 , 614 , 619 , 627 ,	\skip_horizontal:N 240
632 , 636 , 644 , 648 , 654 , 659 , 667 , 680	\skip_horizontal:n 173 , 215 , 527
\l__siunitx_unit_per_bool	\c_zero_skip 174
..... 126 , 294 , 301 , 379 , 390	\sp 132
\l__siunitx_unit_per_symbol_bool	space-before-unit 150
..... 455 , 462 ,	\SplitArgument 67
469 , 476 , 483 , 490 , 508 , 712 , 718 , 764	\SplitList 94 , 103 , 111 , 120
\l__siunitx_unit_per_symbol_tl . .	\square 111 , 971
..... 441 , 799	\squared 111 , 923 , 971
\l__siunitx_unit_position_int . . .	\steradian 110 , 883
..... 130 , 294 , 303 , 306 , 326 , 333 ,	sticky-per 114
344 , 356 , 360 , 370 , 375 , 377 , 382 ,	\stilb 111 , 960
396 , 436 , 522 , 526 , 528 , 534 , 544 , 609	\stokes 111 , 960
\l__siunitx_unit_powers_positive_-	str commands:
bool 456 ,	\str_case_e:nnTF 171
463 , 470 , 477 , 484 , 491 , 508 , 574 , 752	\str_if_eq:nnTF
\l__siunitx_unit_prefix_fp 70 , 152 , 167 , 199 , 285 , 299 , 409 ,
..... 145 , 166 , 515 , 525 , 613	581 , 635 , 657 , 674 , 952 , 1026 , 1072 ,
\l__siunitx_unit_prefix_power_-	1201 , 1245 , 1347 , 1397 , 1440 , 1624
bool 131 , 136 , 514 , 599	\str_if_eq_p:nn 285 , 441 ,
\l__siunitx_unit_prefixes_-	504 , 616 , 618 , 1385 , 1387 , 1548 , 1551
forward_prop 49 , 605	sys commands:
\l__siunitx_unit_prefixes_-	\sys_if_engine luatex_p:
reverse_prop 49 14 , 25 , 42 , 99 , 115 , 821

<code>\sys_if_engine_xetex:TF</code>	304	<code>\textpm</code>	65, 285
<code>\sys_if_engine_xetex_p:</code>		<code>\textsubscript</code>	65, 325, 356
.	15, 26, 43, 100, 116, 822	<code>\textsuperscript</code>	65, 330
T			
<code>table-align-comparator</code>	86	<code>\texttimes</code>	125, 130
<code>table-align-exponent</code>	86	<code>\the</code>	203, 208
<code>table-align-text-after</code>	86	<code>\THz</code>	8, 144
<code>table-align-text-before</code>	86	<code>tight-spacing</code>	25
<code>table-align-uncertainty</code>	86	<code>\times</code>	121, 136, 143, 145, 1649
<code>table-alignment</code>	86	tl commands:	
<code>table-alignment-mode</code>	86	<code>\c_empty_tl</code>	49, 50, 1450
<code>table-auto-round</code>	86	<code>\tl_clear:N</code>	26, 80, 85, 95, 119,
<code>table-column-width</code>	86	121, 124, 125, 126, 143, 144, 144,
<code>table-fixed-width</code>	86	145, 195, 221, 267, 279, 289, 315,
<code>table-format</code>	87	367, 476, 523, 524, 531, 543, 544, 564
<code>table-number-alignment</code>	87	<code>\tl_clear_new:N</code>	83
<code>table-text-alignment</code>	87	<code>\tl_const:Nn</code>	7, 117, 119, 330
<code>\tablenum</code>	145	<code>\tl_count:n</code>	142,
<code>\tabularnewline</code>	54, 56, 75, 77	149, 619, 655, 665, 667, 818, 819,
<code>\tera</code>	13, 58, 109, 860	1037, 1170, 1173, 1180, 1190, 1216,
<code>\tesla</code>	110, 883	1225, 1230, 1247, 1411, 1422, 1492
<code>\TeV</code>	43, 145	<code>\tl_head:n</code>	181,
TeX and L ^A T _E X 2 _ε commands:		235, 309, 917, 935, 1088, 1201, 1478
<code>\@ifpackagelater</code>	4	<code>\tl_head:w</code>	774, 793
<code>\@ifpackageloaded</code>	7, 23, 42,	<code>\tl_if_blank:nTF</code>	3, 17,
.	44, 66, 84, 152, 177, 202, 216, 237, 308	136, 152, 175, 183, 201, 206, 263,
<code>\@maybe@unskip</code>	72	266, 269, 271, 288, 307, 480, 551,
<code>\@temptokena</code>	206, 208	641, 663, 675, 697, 709, 715, 726,
<code>\AtBeginDocument</code>	68	771, 782, 790, 872, 982, 1114, 1155,
<code>\f@family</code>	171	1222, 1335, 1341, 1395, 1475, 1564
<code>\f@series</code>	125	<code>\tl_if_blank_p:n</code>	
<code>\m@th</code>	331, 363	4, 106, 110, 185, 186, 1197, 1386
<code>\math@version</code>	152	<code>\tl_if_empty:N</code>	
<code>\NC@do</code>	201, 202	88, 94, 99, 102, 103, 109,
<code>\NC@find</code>	211	113, 119, 123, 126, 139, 147, 152,
<code>\NC@list</code>	6, 202, 203	162, 196, 222, 227, 230, 257, 270,
<code>\newcommand</code>	151	278, 293, 349, 468, 521, 563, 604,
<code>\protected@edef</code>	15, 66, 86, 106, 121, 193	730, 750, 759, 805, 1320, 1349, 1584
<code>\sf@size</code>	376	<code>\tl_if_empty:nTF</code>	84, 568
<code>\tab@setcr</code>	73	<code>\tl_if_empty_p:N</code>	429, 705, 713
<code>\z@</code>	376	<code>\tl_if_eq:NNTF</code>	137
tex commands:		<code>\tl_if_exist:N</code>	94
<code>\tex_cr:D</code>	31	<code>\tl_if_in:NnTF</code>	5,
<code>\tex_hfil:D</code>	231, 237	57, 116, 216, 275, 306, 381, 387,
<code>\tex_hss:D</code>	257, 259	399, 402, 409, 414, 484, 501, 516, 553
<code>\tex_kern:D</code>	174	<code>\tl_if_novalue:nTF</code>	189, 192
<code>\text</code>	65, 75, 115, 116, 117, 120, 122, 156, 251	<code>\tl_if_single_token:nTF</code>	92
<code>text-family-to-math</code>	66	<code>\tl_map_inline:Nn</code>	236, 346, 387
<code>text-weight-to-math</code>	67	<code>\tl_map_inline:nn</code>	34, 55, 63
<code>\textcolor</code>	65,	<code>\tl_new:N</code>	3,
.	65, 66, 67, 90, 106, 110, 111, 112, 1367	3, 3, 4, 4, 4, 5, 5, 6, 7, 8, 9, 13, 18,
<code>\textminus</code>	65, 68, 289	19, 23, 29, 34, 35, 38, 48, 49, 67,
<code>\textmu</code>	17, 22	68, 69, 70, 71, 72, 73, 74, 81, 82,
		82, 83, 84, 101, 102, 103, 128, 222,

279, 280, 281, 282, 283, 283, 284, 513, 516, 517, 518, 600, 601, 1252, 1576	\unit 67, 155, 166, 174
\tl_put_right:Nn	unit-close-bracket 114
. 14, 46, 58, 60, 81, 140, 141, 150, 153, 154, 157, 308, 378, 389, 431, 443, 470, 486, 506, 522, 627, 679	unit-color 67
\tl_replace_all:Nnn	unit-mode 67
. 3, 6, 11, 11, 111, 181, 183, 210, 213, 238, 242, 301, 327, 343	unit-open-bracket 114
\tl_reverse:n 704, 1009, 1010, 1016	unit-optional-argument 150
\tl_set:Nn 21, 24, 26, 41, 45, 46, 53, 54, 65, 77, 85, 98, 105, 114, 121, 121, 124, 132, 134, 136, 136, 140, 155, 157, 163, 168, 179, 180, 203, 208, 218, 219, 220, 227, 229, 230, 234, 241, 253, 263, 265, 273, 274, 286, 287, 292, 298, 301, 312, 313, 332, 349, 351, 355, 370, 402, 407, 413, 426, 430, 433, 435, 441, 462, 466, 469, 513, 514, 524, 543, 555, 559, 560, 562, 576, 590, 608, 612, 628, 632, 637, 644, 654, 665, 714, 727, 740, 760, 774, 776, 787, 796, 799, 812, 825, 835, 843, 1238	\unitInBookmark 157
\tl_set_eq:NN 47, 54, 69, 92, 153, 215, 263, 277, 498, 577, 593, 619, 807	\unskip 53, 74
\tl_tail:n 775, 794	\upOmega 34, 35
\tl_use:N 33, 74, 95, 109, 143, 216, 232, 276, 315	\upshape 66
\l_tmpa_tl 106, 107	\us 85, 144
token commands:	use commands:
\l_peek_token 336	\use:N 28, 92, 226, 309, 332, 336, 408, 433, 531, 547, 622, 686, 715, 747, 1028, 1413, 1419, 1478
\token_if_eq_charcode_p:NN 557	\use:n 64, 89, 113, 156, 157, 167, 168, 191, 193, 201, 230, 241, 248, 250, 377, 670, 1399, 1442, 1557
\token_if_eq_meaning:NNTF 63, 97	\use_i:nn 77
\token_to_str:N	\use_i:nnnn 115
. 30, 83, 85, 94, 97, 143, 182, 184, 243, 288, 317, 344, 350, 364, 986, 989	\use_i_delimit_by_q_recursion_- stop:nw 1056, 1101, 1626, 1628
\tonne 110, 894	\use_i_delimit_by_q_stop:nw 98
\tothe 112, 115	\use_ii:nn 1205
track-explicit-plus 25	\use_iv:nnnn 107, 111
\ttdefault 175	\use_none:n 480, 1090, 1460
	\use_none:nn 1464
	use-xspace 150
	\uV 21, 145
	\uW 43, 145
	V
	\V 21, 145
	\volt 21, 22, 23, 24, 25, 26, 110, 883
	W
	\W 43, 145
	\watt 43, 44, 45, 46, 47, 48, 59, 110, 883
	\weber 110, 883
	weight-version-mapping 67
	X
	\xspace 88
	Y
	\yocto 109, 850
	\yotta 109, 860
	Z
	\zepto 109, 850
	\zetta 109, 860
U	
\uA 2, 144	
\ug 35, 144	
\uJ 43, 145	
\uL 27, 145	
\ul 27, 48, 145, 151	
\um 61, 144	
\umol 14, 145	
uncertainty-separator 25	